Full Body Interaction

Design, Implementation,
and User Support

Dissertation submitted to the

Faculty of Applied Computer Science

in accordance with the requirements

for the Degree of

Doctor rerum naturalium (Dr. rer. nat.)

by

Felix Kistler

Augsburg University, 2015

Reviewers: Prof. Dr. Elisabeth André
Prof. Dr. Johannes Schöning
Prof. Dr. Jörg Hähner

Date of Examination: 2015-12-15
Examiners: Prof. Dr. Elisabeth André
Prof. Dr. Johannes Schöning
Prof. Dr. Jörg Hähner

Acknowledgments

At first, I want to thank my supervisor, Prof. Dr. Elisabeth André, for her constant support during this dissertation. Her broad interdisciplinary knowledge and her open personality always brought new impulses to my work. Without her guidance, this dissertation would have been impossible. I further would like to thank Prof. Dr. Johannes Schöning for his comprehensive and constructive feedback, and I further thank Prof. Dr. Jörg Hähner for willing to review this dissertation.

I enjoyed working at the lab for Human-Centered Multimedia, and I would like to thank all my colleagues with whom I prepared and held lectures and practical courses, thought about new ideas, implemented and evaluated the ideas, wrote publications, and visited conferences and project meetings together. Similar thanks go out to my colleagues within the eCute project that gave a wonderful basis for many publications and this dissertation as well. I especially want to thank Dr. Ilhan Aslan and Dr. Nick Degens for proof-reading parts of this dissertation. Both colleagues, from the lab and the project, did not only act as professional colleagues, but became real friends during the past years.

I would also like to thank my parents and family for their support, understanding and love throughout my whole life. Last but not least, I especially want to thank my wife Uli: thank you for being so patient, understanding, for constantly motivating me and brightening up my life in the past years. Thank you for your love and for believing in me!

Abstract

While researchers have been working on device-free gestural interaction for multiple decades, full body interaction was first truly introduced to the mass market by the Microsoft Kinect for Xbox 360 in November 2010. Full body interaction is meant as a very unobtrusive and natural interaction modality; users can interact with a computer via motions of the whole body and without touching, wearing or holding any device or special gear. However, there are many differences between full body interaction and traditional interaction technologies such as the mouse and keyboard. It is not easy to provide good usability throughout the interaction because of its lower precision and higher complexity. Thus, multiple challenges still remain until full body interaction will gain further acceptance.

In this dissertation, I investigate those challenges and present three major contributions: In the first, I follow a user-centered design process to create gesture sets that, on the one hand, are intuitive and easy to reproduce for the actual users, while on the other hand, are consistent, unambiguous, and can be recognized with low-cost technology. The second and main technical contribution of this dissertation is the *Full Body Interaction (FUBI)* framework, which can be used to easily integrate full body interaction in arbitrary applications, using an XML-based gesture-definition language that supports powerful gesture recognition. In addition, FUBI can be used to implement freehand interaction with a graphical user interface (GUI) or to implement avatar control. Besides being able to integrate full body interaction in an application, it is also important to support the user during the interaction. The third contribution therefore focuses on mechanisms such as

affordances, feedback and feedforward, to help the users understand which gestures are currently available, how they should be performed, but also why they may not be recognized in certain cases.

In this work, I focus mainly on virtual environments, which are especially suited for full body interaction. To prove the generalizability of my research, I further look at application scenarios in which a user controls GUIs or humanoid robots. Overall, I present concepts, implementations and study results to provide insights on how to improve the process of creating full body interaction applications. I therefore take into account all stake-holders of full body interaction: the interaction designer, the developer, as well as the end user.

Zusammenfassung

Während Wissenschaftler schon seit mehreren Jahrzehnten an gerätefreier Gesteninteraktion arbeiten, wurde Ganzkörperinteraktion erst mit der „Microsoft Kinect for Xbox 360“ im November 2010 für die breite Masse verfügbar. Ganzkörperinteraktion hat das Ziel eine sehr unaufdringliche (engl. „unobtrusive“) und natürliche Interaktionsmodalität zu sein: Nutzer können durch Bewegungen des gesamten Körpers mit einem Computer interagieren, ohne dabei ein Gerät zu berühren, zu tragen oder zu halten. Es gibt allerdings viele Unterschiede zwischen Ganzkörperinteraktion und traditionellen Interaktionstechnologien wie Maus und Tastatur. So ist es insbesondere keine leichte Aufgabe, eine gute Benutzerfreundlichkeit (engl. „usability“) während der Interaktion zu gewährleisten, da die Interaktion grundsätzlich unpräziser, aber auch von höherer Komplexität ist. Somit bleiben noch einige Herausforderungen bis sich Ganzkörperinteraktion weiter durchsetzen kann.

In dieser Dissertation betrachte ich einige dieser Herausforderungen und trage in drei Hauptbeiträgen zu deren Lösung bei: Im ersten Beitrag folge ich einem nutzerzentrierten Designprozess, um Gestensätze zu entwerfen, welche zum Einen intuitiv und leicht für die Nutzer zu verstehen und zum Anderen konsistent, eindeutig und gut mit kostengünstiger Technologie erkennbar sein sollten. Der zweite und aus technischer Sicht wichtigste Beitrag dieser Dissertation ist das *Full Body Interaction Framework (FUBI)*, eine Software, welche es ermöglicht, Ganzkörperinteraktion ohne großen Aufwand in beliebige Anwendungen zu integrieren. Dazu ver-

wendet FUBI eine XML-basierte Gestendefinitionssprache, die trotz ihrer Einfachheit eine leistungsfähige Gestenerkennung ermöglicht. Zusätzlich erlaubt es FUBI, Freihand-Interaktion mit einer grafischen Benutzeroberfläche oder eine Avatarsteuerung umzusetzen. Allerdings ist es nicht damit getan, Ganzkörperinteraktion in eine Anwendung zu integrieren. Es ist genauso wichtig, die Nutzer bei der Interaktion zu unterstützen. Der dritte Hauptbeitrag widmet sich folglich Mechanismen wie Affordanzen, Rück- und Vormeldungen (engl. „affordances, feedback and feedforward“), um den Nutzern aufzuzeigen, welche Gesteneingaben gerade zur Verfügung stehen, wie diese ausgeführt werden, aber z. B. auch, wieso eine Geste in manchen Fällen nicht erkannt worden ist.

Ich konzentriere mich in dieser Arbeit hauptsächlich auf virtuelle Umgebungen, die besonders für diese Art der Interaktion geeignet sind. Um trotzdem Generalisierbarkeit zu gewährleisten, betrachte ich weiterhin Anwendungsfälle, in denen grafische Benutzeroberflächen oder Roboter gesteuert werden. Insgesamt präsentiere ich verschiedene Konzepte, Implementierungen und Studienergebnisse, die Erkenntnisse bringen, wie man den Prozess der Erstellung von Anwendungen mit Ganzkörperinteraktion verbessern kann. Ich berücksichtige dabei alle Beteiligten: den Interaktionsdesigner, den Entwickler und den Endnutzer.

Publications Related to the Dissertation

Parts of the ideas, figures, data, and descriptions from this dissertation have already been published. In the following, I provide a list of corresponding publications and briefly summarize how they are related to this thesis. In case I am not the first author of a publication, I briefly describe my contributions to it.

Peer-Reviewed Conference Publications

Felix Kistler and Elisabeth André. User-defined body gestures for an interactive storytelling scenario. In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2013*, volume 8118 of *Lecture Notes in Computer Science*, pages 264–281. Springer Berlin Heidelberg, 2013.

Basis for Section 4.3 and 7.4; application of the approach for user-defined body gestures in the intercultural training application Traveller; introduction of the XML-based gesture definition language, the taxonomy of Section 4.1.2, and the onscreen gesture symbols of Section 7.4.1.

Felix Kistler, Dominik Sollfrank, Nikolaus Bee, and Elisabeth André. Full body gestures enhancing a game book for interactive story telling.

In Mei Si, David Thue, Elisabeth André, James Lester, Joshua Tanenbaum, and Veronica Zammitto, editors, *Interactive Storytelling*, volume 7069 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin Heidelberg, 2011.

Basis for Section 3.1; first mention of the FUBI framework; first version of onscreen gesture symbols.

Peer-Reviewed Journal Articles

Felix Kistler, Birgit Endrass, Ionut Damian, Chi Tai Dang, and Elisabeth André. Natural interaction with culturally adaptive virtual characters. *Journal on Multimodal User Interfaces*, 6:39–47, 2012.

Basis for Section 5.2 and 7.1; first comprehensive description of the FUBI framework with gesture recognition and avatar control.

Mohammad Obaid, **Felix Kistler**, Markus Häring, René Bühling, and Elisabeth André. A framework for user-defined body gestures to control a humanoid robot. *International Journal of Social Robotics*, pages 1–14, 2014.

Basis for Section 4.2 and 7.3; first mention of the finger count recognizers. I contributed to the paper by helping at the design of the study for user-defined gestures. I conducted this study together with Markus Häring, and together with the other authors, I ran the analysis and summarized the results. I wrote the "Implications for Gesture Recognition" and implemented the recognition and evaluation software for the follow-up study. I was not involved in the conduction of the second study, but I again helped at the analysis and the writing of the results.

Peer-Reviewed Poster, Demo and Workshop Publications

Felix Kistler, Birgit Endrass, and Elisabeth André. Full body interaction with virtual characters in an interactive storytelling scenario. In Timothy Bickmore, Stacy Marsella, Candace Sidner, editors, *Intelligent Virtual Agents*, volume 8637 of *Lecture Notes in Computer Science*, pages 236–239. Springer Berlin Heidelberg, 2014.

Evaluation results of Section 7.4.3.

Felix Kistler, Elisabeth André, Samuel Mascarenhas, André Silva, Ana Paiva, Nick Degens, GertJan Hofstede, Eva Krumhuber, Arvid Kappas, and Ruth Aylett. Traveller: An interactive cultural training system controlled by user-defined body gestures. In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2013*, volume 8120 of *Lecture Notes in Computer Science*, pages 697–704. Springer Berlin Heidelberg, 2013 **(Best Poster Award)**.

Additions for Section 4.3 and 7.4; introduction of the swipe menu for the selection of dialogue phrases.

Felix Kistler and Elisabeth André How can I interact? Comparing full body gesture visualizations. In *Proc. CHI PLAY 2015*, pages 583–588. ACM, 2015.

Basis for Chapter 6, and in particular Section 6.2.

Kathrin Janowski, **Felix Kistler**, and Elisabeth André. Gestures or speech? Comparing modality selection for different interaction tasks in a virtual environment. In *Proc. Tilburg Gesture Research Meeting 2013*. http://tiger.uvt.nl/pdf/papers/janowski.pdf (accessed 2015-9-15), 2013.

Basis for Section 3.2; comparison of the FUBI gestural interaction with speech input; first investigation of the mapping between in-game actions and gestures. I supervised Kathrin Janowski during her

Master thesis, which formed the basis for this publication. Kathrin Janowski contributed the main part of the implementations and the conduction of the study, but I helped her with the integration of the FUBI software, the literature research and the study design. The study analysis and the writing of the paper was cooperative work with roughly equal contributions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the last decade, the way humans interact with computers became much more gestural. While in the early days of personal computers, text input via keyboard was mainly used to control command line-based operating systems, the computer mouse soon paved the way for graphical user interfaces (GUIs). Today, both – the mouse and the keyboard – are still the most common input devices in office workplaces.

Personal digital assitants (PDAs) started a first attempt to bring gestural interaction into daily life in the 1990s by providing touch interaction with a stylus, but it became truly popular with the introduction of multi-touch smartphones, such as the Apple iPhone[1] that was released in 2007. Those smartphones offered interaction according to the principle of direct manipulation, which they shared with other touch-enabled devices ranging from small watches to huge interactive tables or even whole walls.

In parallel, a different interaction modality was propagated by devices for motion-based interaction such as the Nintendo Wii Remote[2] in 2006, which applied accelerometers together with IR tracking. This kind of interaction let the user move away from the screen and use a larger room in

[1] http://apple.com/iphone/ (accessed 2015-9-15)
[2] http://wii.com (accessed 2015-9-15)

front of it. However, users would still need to hold a device in their hands.

Figure 1.1: Microsoft Kinect for Xbox 360[3]

The next step for making human-computer interaction more gestural in daily life can again be attributed to a device mainly used for entertainment gaming: the Microsoft Kinect for Xbox 360[4] (see Figure 1.1), which I will further refer to as "Kinect". The Kinect's main input device was a depth camera, which eased to track the shape and posture of persons in its field-of-view and allowed users to interact without holding any devices, but rather through motions of their whole body. It should also be mentioned that the Kinect included a microphone array to provide speech input, but I will only focus on its full body interaction capabilities in this dissertation. The Kinect was the first depth sensor providing sufficient accuracy (random error < 1 cm and depth resolution < 2 cm within its main range of interaction [89]) with an easy setup and configuration, at a price low enough to be available for the average consumer (≈ 150 Euros at its launch). While this novel kind of input device offered a lot of new interaction possibilities, it also brought new challenges for the interaction designer, as it was very different with regard to traditional interaction technologies. For this reason, the designers of Kinect games have had difficulties to provide good usability and, as a result, many game concepts failed to engage the users.

In this dissertation, I will investigate ways to improve the process of integrating full body interaction in virtual environments and some other application scenarios. My contributions range from the design of gesture sets through a user-centered approach, to defining and recognizing input gestures via the implementation of the full body interaction (FUBI) frame-

[3] *Image source:* http://commons.wikimedia.org/wiki/File:Xbox-360-Kinect-Standalone.png (accessed 2015-9-8)

[4] http://support.xbox.com/browse/xbox-360/kinect (accessed 2015-9-15)

work, and finally to the support of users during the interaction by using affordances, feedback and feedforward mechanisms.

1.2 Concepts and Terms

In the following, I briefly define important concepts and terms as used in the rest of the thesis:

- **Device-free Interaction:** (Human-computer) interaction without holding a device in the hands, wearing special gear, or touching a device in general. This kind of interaction is accordingly considered as especially unobtrusive. It usually requires a camera that is able to track the users' actions.
- **Freehand Interaction = Hands-free Interaction = Midair Interaction:** Same as device-free interaction, but only targeting the hands. In other words, interaction using hand movements, but without holding devices in the hands, wearing gloves, or touching a device with the hands in general.
- **Gesture:** Intentional movement of the hands, other body parts or the whole body, which can be used as input for a computer. For example, waving is a gesture involving the arms, walking in place is a gesture involving the legs, and jumping is a gesture involving the whole body (although the gesture is initiated by the legs).
- **Posture:** A specific configuration of body parts, but again with the intention to be input for a computer. For example, crossed arms are a posture involving the arms, thumbs up is a hand or finger posture, a squatting position would be a posture involving the legs. In the context of this dissertation, postures will sometimes be referred to as a special form of gestures without movement.
- **Gestural Interaction = Gesture-based Interaction:** (Human-Computer) Interaction via gestures and postures. Can involve the whole body or hands only, and can require a hand-hold, a worn or touch-device or can be device-free.

- **Full Body Interaction = Whole Body Interaction:** Gesture-based interaction involving motions of the whole body. For example, users can give input to a computer through: arm movements, leg or head movements, a specific body posture, turning their body, moving in a certain direction, or jumping.
- **Depth Sensor = Depth Camera:** Camera capturing and providing a depth image of its field-of-view.
- **Depth Image:** Camera image in which the color value of each pixel represents the distance of the object hit by the corresponding projection ray from the camera.

1.3 Application Scenarios

In many present applications, interaction is still largely determined by traditional input devices, such as the mouse and keyboard. However, the Kinect established device-free full body interaction in the **gaming** sector, in which it is particularly used for sport, fitness, and party games. Other depth sensors such as the LEAP Motion Controller[5], the Asus/PrimeSense cameras[6], or the Intel RealSense[7] followed, which started to spread this new kind of interaction for **home entertainment systems** and **arbitrary computers**. In this section, I will describe application scenarios in which I see further potential for my research (cf. Mitra and Acharya [125] or Karam and Schraefel [85]). The continuum of gestural input by Kurtenbach and Hulteen [106], ranges from redundant information in parallel to another modality to the primary channel of input. According to the continuum, this dissertation will focus on the far end. I will regard gestural input solely as the primary channel of input, and – with the exception of Section 3.2 – I will not look at other modalities or investigate multimodal input.

Full body interaction is a form of device-free interaction, and in general, this is a good choice in scenarios, in which it is **undesired or impossible**

[5] http://leapmotion.com (accessed 2015-9-15)
[6] http://asus.com/3D-Sensor/Xtion_PRO (accessed 2015-9-15)
[7] http://intel.com/realsense (accessed 2015-9-15)

to touch anything. For example, a surgeon could control an endoscope with freehand gestures during a surgery or a car mechanic could control the diagnosis software with gestural commands while working on a car's engine. A less specialized application is the interaction with **public displays** in general (cf. Figure 1.2) and in particular in **museum or art installations**. In most cases, users spontaneously interact with those systems and so it is easier for the users if they do not have to pick up a device, first. Furthermore, it may not be appropriate to use a touch screen, as the screen is too large or out of the users' range, or full body interaction itself is used to attract the interest of users that have not used it before. Even if a touch screen is present, full body interaction can still be a good option to enhance an interactive experience (cf. [127]).

Figure 1.2: Child engaged in freehand interaction with a public display

Another application area for full body interaction is given by **virtual environments** and especially in combination with **virtual or augmented reality** (VR or AR respectively). Therein, the goal is to create a highly immersive interaction for the user. Full body interaction seems to be a promising option for close-to-real-life interaction. Users perform body gestures that directly represent the actions they would do in real life, instead of triggering those actions in an abstract manner by pressing buttons.

Full body interaction is also useful when the application actually needs

to **analyze body postures and motions** of the users, for instance in dance learning applications, in rehabilitation or fitness games, or in learning applications that include non-verbal behaviors, such as job application exercises or intercultural training. Similar advantages are present in applications in which **objects have to be manipulated in three dimensions**, such as 3D modeling, animation or sculpting.

While full body interaction can be used to control any device or machine, I believe that it is especially suited for controls that mirror real-life behavior. For example, a **humanoid robot** is usually designed to look and behave like a human. Therefore, it should be most suitable to communicate with it in a real-life manner, e.g. using pointing gestures to indicate navigation targets or directions.

For other interaction tasks, such as controlling a GUI or entering text, it is usually not the best idea to apply full body interaction, as those tasks are usually performed better with a different modality, e.g. a mouse or a keyboard. Nevertheless, there are scenarios, in which it is worth conducting them with full body interaction, despite the lower performance. One such scenario is given by **tasks during a motion-based game apart from the gameplay mechanics**, e.g. changing game settings or entering a name in a high score during a full body interaction game. In current full body interaction games, e.g. those on the Microsoft Xbox consoles using the Kinect, the mentioned interaction tasks often require to use a different modality, e.g. a game controller. Switching modalities usually takes some time, as the user may have to pick up a game controller, and that will interrupt the game experience. It can even lead to unwanted behavior of the game that tries to interpret the motion of picking up the controller as an input gesture. Therefore, it should be a better choice to stay within the full body interaction modality. In the example above, this is further amplified by the fact that the game controller itself is not perfectly suited for text entry or the interaction with a GUI.

1.4 Challenges

Although device-free full body interaction opens up many new possibilities for traditional human-computer interaction, it also presents several challenges for the interaction designer, the developer, and the end user, which will be further investigated in this dissertation.

Regarding the interaction design, well-known paradigms and interface guidelines often do not apply to full body interaction. For example, in a GUI controlled with a mouse cursor, it can be a good idea to use complex menu structures to organize the interaction options. However, when trying to port that paradigm to full body interaction, it can get almost impossible for the user to control the cursor precisely enough for navigating through the now-too-complex menu. In general, the interaction requires more space, should be less fine-grained, and is in most cases designed for users standing in front of a larger screen. Practitioners are often not used to the theories of human non-verbal behavior (e.g. the theories by McNeill [121]) or physiological restrictions, which makes it hard for them to categorize and compare gestures or to estimate their usability. Furthermore, gesture sets are often created according to the opinions and preferences of their developers, or by simply looking at what gestures can be recognized best and do not conflict with each other. However, the resulting gestures might not be intuitive for the actual users.

Full body interaction further poses challenges from an engineering point-of view. Traditional gesture recognition algorithms are often optimized for a single stream of two-dimensional points, in which the start and end of a gesture is known because of a manual segmentation. In some scenarios, e.g. mouse interaction, users already hold a device in their hands, and can press a button to mark the start and end of a gesture explicitly. In other scenarios, e.g. touch interaction, the segmentation is even determined implicitly by the start and end of a touch on an interactive surface. In opposite to those scenarios, device-free full body interaction incorporates multiple streams of three-dimensional points (all tracked joint positions and orientations) and there is no straight-forward way for manual data segmentation. Ideally, such a segmentation should happen automatically by the system, but if manual

segmentation is required, it should at least be unobtrusive and integrated with the actual gesturing in a natural way. Another technical challenge that is present in all current full body interaction applications, both in research as in the industry (e.g. games with Kinect interaction on the Microsoft Xbox console), is the low quality of the data. First, it is already hard for the user to perform an accurate gesture in midair. Secondly, the depth image construction produces data with a certain level of noise in it. Finally, additional noise is created during the user tracking and the tracking can even fail completely from time to time. The interaction can therefore get quite unstable and inaccurate. Especially for complex interaction tasks, this can severely frustrate the users. Simplifying the interaction might not be a good choice as well, as it can make an application less powerful or make a game boring. Therefore, developers need to strike the right balance between an expressive interaction and one that is easy-to-use.

There are also challenges for the end user of the full body interaction application. In general, gestural interaction is still quite novel to computer users and there is no common standard gesture set for full body computer interaction. One has to suppose that the users do not know which system effects they can trigger with which actions. There have not yet emerged common guidelines on how to apply supporting mechanisms such as feedback and feedforward, which for example raises the questions: how can users be informed what gestures are available at a certain point in time and how can they learn to perform them? How can the system guide the user while performing a gesture and how can it detect that the user tried to perform a certain gesture, but failed?

1.5 Research Goals

The overall goal of this thesis is to find improvements for the process of creating full body interaction applications, while addressing the challenges of the preceding section and considering all relevant stakeholders: the interaction designers, the application developers, and the end users. This is done according to the following five steps:

1. **Exploration of Full Body Interaction.** In this first step, I will first present multiple concepts of full body interaction. I will compare different full body interaction types, i.e. full body gestures and freehand GUI interaction, and I will briefly compare full body interaction with the related speech modality.

2. **User-defined Full Body Gestures.** In the second step, I will first provide tools for categorizing and measuring full body interaction and gesture sets. This forms the basis for the actual gesture design, in which I will investigate how to create gesture sets with an intuitive mapping to the system events they are meant to trigger, following a user-centered design approach.

3. **The Full Body Interaction Framework.** The third step will target the developers of full body interaction applications and I will present the FUBI framework. FUBI aims to provide an easy way for implementing full body interaction and supporting powerful gesture recognition, as well as freehand GUI interaction, and avatar control. I will focus on full body gesture recognition and multiple recognition techniques will be investigated.

4. **Supporting Full Body Interaction Users.** The fourth step focuses on the integration of full body interaction in the application and the role of the end user. I will investigate ways to support users during full body interaction in terms of letting them know whether and how they can interact, what gestures are currently available, how they should be performed, and also whether their gesture performance is recognized or why it stays unrecognized in certain cases.

5. **Evaluations.** In the last step, I will present multiple evaluation studies that are meant to validate the work presented in the preceding steps. I will evaluate avatar control and freehand GUI interaction, each with one sample application. I will further implement the gesture recognition and initial user support for the two user-defined gesture sets and evaluate their user experience and recognition accuracy.

In conclusion, step 1 serves as an introduction, illustrates the relevance

of full body interaction and forms the basis for the later developed taxonomy of full body interaction. Steps 2–4 form the main contributions of the dissertation and target the three stakeholders of full body interaction (designer, developer, and end user). Step 5 further validates the statements and contributions of the dissertation.

1.6 Organizational Overview

The rest of the thesis is structured as follows:

In Chapter 2, I will provide a theoretical background and investigating previous work related to the topics of this thesis. This will include potential hardware and tracking software in Section 2.1, previously defined taxonomies and coding schemes potentially suitable for full body interaction in Section 2.2, algorithms and frameworks for posture and gesture recognition in Section 2.3, technologies for freehand GUI interaction in Section 2.4, methods for creating user-defined gesture sets in Section 2.5, methods for guiding the user during the interaction by providing affordances, feedback and feedforward signals in Section 2.6, and several applications that are suited for integrating full body interaction in Section 2.7.

The next five chapters describe my own contributions, structured according to the formulated research goals listed in the previous section:

In Chapter 3, I will explore full body interaction. I will compare different full body interaction types, as well as the relation between gestures and speech according to a range of common interaction tasks that are usually found in virtual environments.

In Chapter 4, I will investigate the design of user-defined gestures sets. Therein, I will first define taxonomies for full body interaction in Section 4.1, to be able to categorize the interaction. In Section 4.2 and 4.3, I will investigate how to create intuitive full body gestures according to the method introduced by Wobbrock et al. [189].

In Chapter 5, I will look at the implementation of full body interaction. At first, I will describe the development of our full body interaction framework (FUBI). The subsequent sections describe the different interac-

tion types that can be implemented using FUBI: full body avatar control in Section 5.2, freehand GUI interaction in Section 5.3, and full body gesture recognition in Section 5.4.

In Chapter 6, I will investigate how to support the user during full body interaction. Therefore, I will first describe the options available in the FUBI framework, to implement affordances, feedback, and feedforward mechanisms with examples of the applications presented in this thesis. Furthermore, in Section 6.2 I will depict a study that compares how the visualization techniques of gesture symbols influence the interaction with a full body interaction application.

In Chapter 7, I will present multiple evaluation studies, conducted to validate the findings of Chapters 4–6. The studies target full body avatar control, freehand GUI interaction, and the user-defined gesture sets.

At the end of the dissertation, I will draw conclusions, summarize the contributions of my dissertation, and look at possibilities for future research in Chapter 8.

Chapter 2

Background and Related Work

In this chapter, I describe different technologies for gesture-based interaction, and for full body interaction in particular. I further provide background on how to categorize and recognize gestures and summarize existing full body interaction frameworks. After that, I investigate related work on freehand GUI interaction, user-defined gesture sets, and mechanisms for supporting full body interaction users. At the end of the chapter, I present several application scenarios for full body interaction in more detail.

2.1 Gesture-based Input Technologies

For applying gestural human computer interaction, different technologies can be used to let the computer track the users' motions. Depending on the technology, users have to touch a surface, hold a device in their hand, shift their weight on a pressure mat, or perform the gestures in midair. Further, different techniques are needed to recognize the gestures out of the provided data streams. In this section, I give a brief overview on the most common technologies and techniques for gestural interaction. As an introduction, Table 2.1 corresponding technologies:

Table 2.1: Technologies for gesture-based input

Technology	Devices	Interaction
surface-based interaction	mobile devices, touch displays, tables, and walls	motions of (mainly) 2D touches on a surface with a stylus, the hands, or tangible objects
device-based interaction	handheld or wearable devices and pressure mats or boards	2D/3D motions of the devices; physical buttons; weight displacement; bio sensors
camera-based interaction	color, infrared, or depth cameras; opt. passive or active markers	motions of body parts or objects such as markers

2.1.1 Surface-based Interaction

An early technology that enabled gestural interaction, are interactive surfaces. Users can interact by touching the surface with their hands or with a dedicated device, such as a stylus. Changing the touch points on the surface,

Figure 2.1: Microsoft PixelSense[1] table with tangible interface objects[2]

[1] http://microsoft.com/en-us/pixelsense/default.aspx (accessed 2015-9-15)
[2] *Image source:* http://commons.wikimedia.org/wiki/File:Surface_table.JPG (accessed 2015-9-8)

results in input to the computer. The technology is integrated in various devices, ranging from small handheld or worn devices to large interactive tables (see Figure 2.1) and walls. The advantages of this technology are that it provides direct manipulation and current devices support multiple touch points in parallel. Disadvantages are that the interaction is limited or at least strongly related to a two dimensional surface and the focus is usually on hand and finger movements.

2.1.2 Device-based Interaction

Other technologies for gestural interaction employ a dedicated interaction device that has to be hold, worn, or sometimes stood on. Therefore, users are not completely free in performing gestures, but the devices provide improved accuracy or can simplify the recognition of inputs.

Handheld Devices

The traditional computer mouse is a handheld device that can be used for gestural interaction. However, it requires the interaction to be conducted on a relatively limited space and in general only supports two-dimensional gestures. More interesting handheld devices for gestural interaction include motion sensors that are used for the main interaction. Nevertheless, they usually own additional physical buttons as the traditional mouse. Motion sensors can measure position (GPS or LED/marker tracking), acceleration (accelerometer), orientation (gyroscope, magnetometer or LED/marker tracking), posture, weight and weight displacements (pressure sensors), or height (air pressure sensor). A typical handheld device for gestural interaction is the Nintendo Wii Remote (see Figure 2.2). Such handheld devices provide the possibility to use three dimensional and larger motions compared to surface-based interaction. Although most handheld devices provide multiple data streams according to their different integrated sensors, those streams usually include only single point data, e.g. a Nintendo Wii Remote only provides a single position/orientation for the hand it holds. Similar to surface-based interaction, such devices focus on hand movements.

Figure 2.2: Nintendo Wii Remote game controller[3]

Wearable Devices

Wearable devices are quite similar to handheld devices and usually include similar motions sensors. However, instead of holding the device in the hands, it is somehow attached to the body, for example by including it in clothing as gloves or shirts, or by mounting it on the body with straps. In Figure 2.3, you can see two examples of wearable interaction devices: a data glove and a motion capturing suit. While the first one mainly measures the angles of all finger joints, the latter provides angles for the main body joints as well as positional data and global orientations. Bolt's famous "Put-That-There" interface [18] includes another example of a wearable interface. In this early example, the wearable device needed an additional stationary device to which it could be relied to and only provided information about the pointing direction of one arm. Apart from tracking joint positions and orientations, wearable devices can also be used to measure bio signals such as the heart rate, skin conductance, or muscle activity. Advantages of wearable devices are that they are often designed to be less obtrusive than handheld ones and that they can track additional body points instead of

[3] *Image source:* http://commons.wikimedia.org/wiki/File:Wii_Remote_Image.jpg (accessed 2015-9-8)

only the hands. Therefore, they allow real full body interaction, although users can still be influenced by the devices on their body and the setup of the devices can require some time. It is further less common to have buttons on a wearable device than on handheld ones.

Figure 2.3: *Left*: P5 data glove[4]; *Right:* Xsens MVN motion capture suit based on intertial sensors[5]

2.1.3 Camera-based Interaction

Another option for applying gestural interaction are cameras. In this way, users do not have to touch a device at all, but their actions are recognized by computer vision. Camera-based solutions can track multiple body points and multiple users at once. However, especially marker-less camera tracking is usually less accurate than device-based tracking and one major issue is the so-called Midas Touch problem. As there are no buttons to press in this kind of interaction, there is no straightforward way to distinguish whether a body movement is meant as an input to the system or not.

[4] http://cwonline.com/store/view_product.asp?Product=1179 (accessed 2015-9-8); *image source:* http://commons.wikimedia.org/wiki/File:P5_in_use.jpg (accessed 2015-9-8)

[5] http://xsens.com/products/xsens-mvn (accessed 2015-9-15)

Cameras Types

There exist different camera types depending on the wavelength range they capture or the information they provide. Color cameras capture images in a wavelength range similar to human vision, but usually with the important difference, that no depth perception is given (monocular vision). They can be appropriate to track objects with a specific color, e.g. hands/faces with skin color, or color patterns, e.g. QR codes. Depth information for tracked objects can be obtained with triangulation when using a stereo camera pair.

Another important camera type are infrared cameras that capture a wavelength range out of human vision. This can be useful to track objects that are marked with a color that should not be seen by humans or does not occur often in the environment. The effect can be emphasized by using infrared reflective materials that are irradiated by infrared LEDs.

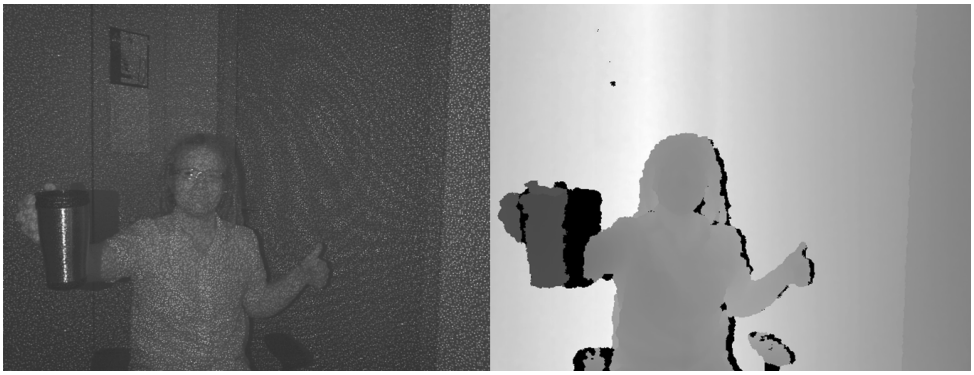


Figure 2.4: Depth image construction using the structured light principle

According to the structured light principle, infrared cameras can further be used to capture an infrared pattern invisible to the human eye that is projected onto the environment. By investigating the distortion of this pattern, distances from a part of the environment to the camera can be estimated as it is done by the first generation Kinect (see Figure 2.4). This also leads to another type of cameras: depth cameras. Depth cameras provide depth perception with an image, in which every pixel describes the distance from the camera to the object hit by the corresponding projection ray (see Figure 2.4 on the right-hand side). There are different ways to capture such a depth image.

Besides the aforementioned structured light principle, another important way to capture the depth image is the time-of-flight priniciple, which was the de facto standard before the release of the Microsoft Kinect for Xbox 360. Time-of-flight cameras measure the time it takes for a light signal to return from an object it hits to the camera. Based on the known speed of light, the distance to the object can be calculated. This principle is used in the second generation Kinect (Microsoft Kinect for Xbox One[6] and for Windows v2, see Figure 2.5). Of course, the two presented

Figure 2.5: Microsoft Kinect for Xbox One[7]

depth camera types have existed before the release of the Kinect. However, they never have provided a similar high resolution while being sold at such a low price and in such large amounts. Before the Kinect, technology that supported similar features cost many times the price of the Kinect while requiring a complex setup and configuration and often even providing lower accuracy. This is why such technology was only available for research prototypes. On the other hand, previous consumer products, such as the Nintendo Wii Remote, only provided much simpler data and interaction as described above. Only the Kinect made depth cameras and corresponding full body interaction available for every home. A third type of depth camera uses a stereo-camera pair and analyzes the two camera images to construct

[6] http://xbox.com/kinect (accessed 2015-9-15)

[7] *Image source:* http://commons.wikimedia.org/wiki/File:Xbox-One-Kinect.jpg (accessed 2015-9-8)

a depth image by matching image parts and triangulation, similar as it is done in human stereoscopic vision. Often, stereo-cameras additionally use LEDs or other light sources to illuminate the scene with pattern-less (IR) light. An example of a stereo-camera depth sensor is the LEAP Motion Controller. The three mentioned depth camera types are summarized with examples, advantages and disadvantages in Table 2.2 (cf. [12, 89, 108]):

Table 2.2: Types of depth cameras (*Abbreviations:* res. = resolution; accy. = accuracy; dist. = distances; esp. = especially)

Camera Type	**Examples**	**Advantages**	**Disadvantages**
Structured-light	Microsoft Kinect for Xbox 360 / Windows v1, Asus Xtion PRO, PrimeSense Carmine, Intel RealSense F200	high image res., high depth res. for lower dist., low cost	shadow artifacts, problems w/ dull, shiny, small, or under-sharp-angle surfaces, low accy. esp. at farther dist., low depth res. at farther dist.
Time-of-flight	Microsoft Kinect for Xbox One / Windows v2, SoftKinetic DS325, Mesa Imaging SR4000, Bluetechnix Argos^{3D} P320	high accy., depth res. & frame-rates	low image res., high cost, problems w/ sharp edges, motion artifacts, high power requirements
Stereo camera triangulation	LEAP Motion Controller, VisLab 3DV-A, Stereolabs ZED	high frame-rates & image res.	low accy. esp. for farther dist. & regions w/ homogeneous color, often no complete depth image

All three depth camera types usually capture IR light to be unobtrusive. They all capture a depth image of their whole field-of-view at a certain point in time and with a frame-rate high enough to capture common human motions. Especially the cameras using structured-light are mostly suited for

indoor use, as they can be disturbed by sun light or other IR light sources. They can further have problems with materials that reflect or absorb the used IR light range in an extreme way. Apart from those three depth camera types, there exist several other approaches such as modulated phase-shifting [28] or triangulation with multiple laser beams [141], however, the three mentioned types are the most commonly found depth camera types for full body interaction, while the others are still in development, not suitable for full body interaction, or not available for end consumers in general.

Different camera types can be combined to benefit from their different advantages. For this purpose, it is helpful to register the viewing frustums of the used cameras to know, e.g. which depth pixel belongs to which color pixel and vice versa, when using a setup with a depth and color camera. This technique is also used by the Kinect.

User Tracking

Figure 2.6: *Left:* IR reflective markers attached to legs[8]; *Right:* Face detection in a color image[9]

One can distinct between marker-based and marker-less user tracking. Markers are objects that have a distinct shape or material properties (infrared reflective, special color or visual pattern), which makes them easy to

[8] *Image source:* http://commons.wikimedia.org/wiki/File:Kistler_plates.jpg (accessed 2015-9-8)

[9] *Image source:* http://commons.wikimedia.org/wiki/File:Face_detection.jpg (accessed 2015-9-8)

track in image streams for computers. Therefore, the computer first knows the two dimensional position of the marker in the camera image. Under certain conditions, it can further be possible to calculate the orientation or distance of the object, based on how it is warped in the camera image. When markers are placed on specific body parts, the motions of those body parts can be tracked and used for interaction (see Figure 2.6 on the left-hand side). Without markers, computers can still track important parts of a color image, e.g. by detecting skin color (see Figure 2.6 on the right-hand side). However, in this case it is less easy to calculate distances, as the actual size of tracked objects (*here:* faces) is unknown and can only be roughly estimated. When using a a stereo-camera pair, triangulation can be applied to calculate the distance of image regions that have been matched between the two cameras [30].

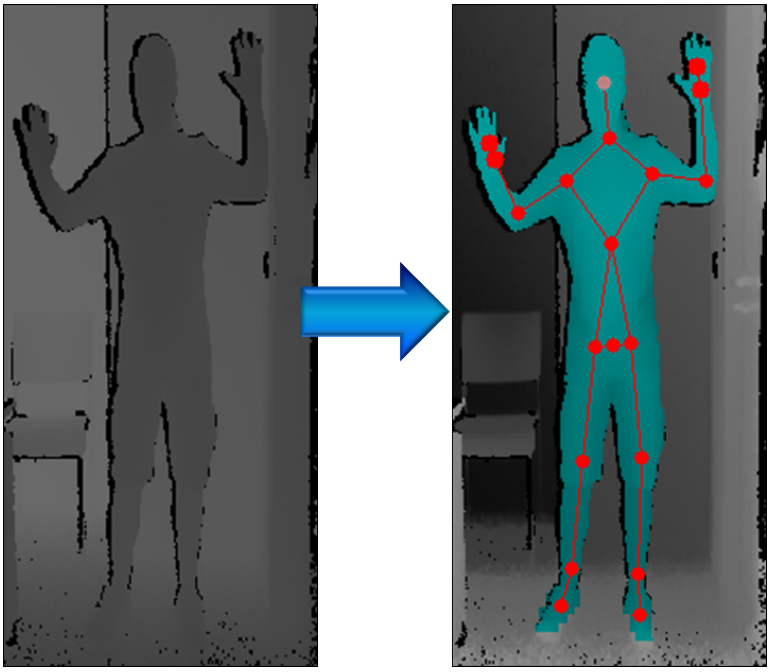


Figure 2.7: User tracking in a depth image

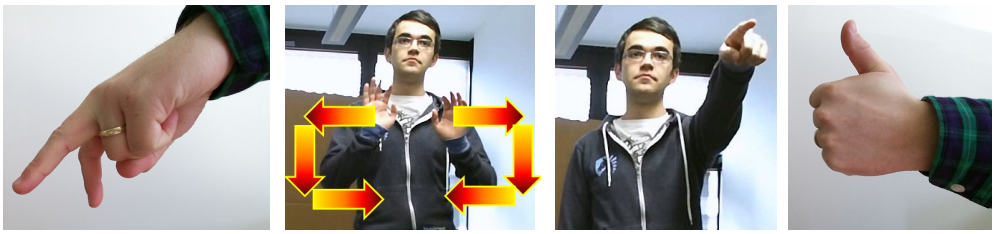
Depth images make it easier to recognize three dimensional shapes in the image, which allows to track persons in the field-of-view and their specific joint configurations (see Figure 2.7). Depending on the technology, different information is provided by the tracking system. The most basic

information is the position of different body parts, either two dimensional in the camera image, or three dimensional in real world space. Furthermore, the orientation of certain body parts may be provided, which can be calculated out of the positional information under certain conditions, e.g. for intermediate joints. One frame of user tracking data describes the current configurations of users' body parts, which need to be interpreted to recognize certain postures. The differences of those configurations between two or more frames describe the movements of users' body parts, and need to be interpreted to recognize gestures.

2.2 Taxonomies and Coding Schemes

Before interpreting body configurations and movements to recognize postures and gestures, it is important to know which kind of human postures and gestures can be observed. There exist different taxonomies for the classification of conversational human gestures in the social sciences. One of the first was introduced by Efron [43] who presented five categories: physiographics, kinetographics, ideographics, deictics, and batons. Ekman and Friesen [44] tried to classify gestures on a functional level with the three categories emblems, illustrators, and manipulators. Furthermore, Kendon [88] tried to link his gesture taxonomy to the relation with speech and defined the following categories: gesticulation, language-like gestures, pantomime, emblems, and sign language. Another popular taxonomy was proposed by McNeill [121] who presented five types of gestures: iconic, metaphoric, deictic, emblematic, and beat gestures. The different taxonomies have a considerable overlap in their covered concepts, and they all focus on hand gestures during a conversation. However, the properties they describe can also be used to categorize full body gestures for human computer interaction. Unfortunately, none of the mentioned taxonomies perfectly suits this purpose. For this reason, and for not confusing the reader, I only describe the items of the terminology by McNeill [121] in more detail in the following.

Iconic and metaphoric gestures both try to convey information by visually depicting an icon. However, they do this at different abstraction



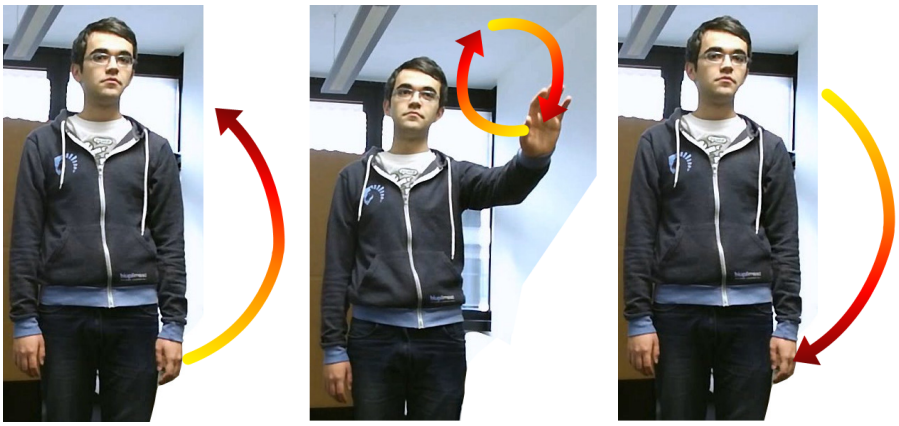
(a) Iconic (b) Metaphoric (c) Deictic (d) Emblematic

Figure 2.8: Sample gestures for McNeill's taxonomy

layers: iconic gestures are more concrete and directly represent a physical, spatial, or temporal property of a real-world referent, e.g. when moving two fingers to indicate somebody is walking (cf. Figure 2.8a). Metaphoric gestures refer to abstract properties of a referent. For example, somebody might form a container with the hands to refer to the contents of a story (cf. Figure 2.8b). Deictic gestures are pointing gestures that indicate a position or direction (cf. Figure 2.8c). Emblematic gestures – in opposite to the other categories – do not accompany speech, but replace speech and are therefore "unspoken words". Emblems are part of a social code and are strongly culture specific. An example is thumbs-up (cf. Figure 2.8d), which has a positive meaning in most western countries, but negative meanings in others. Beat gestures do not convey meaning but accompany speech to emphasize parts of it. Therefore, they are less suited for direct interaction and are neither included in Figure 2.8, nor in my own taxonomy. The other four categories seem to be good candidates for representing input gestures and that is why I will use them in my later described taxonomy.

McNeill [122] also defined three phases of a gesture: preparation, stroke, and retraction. Preparation is the phase in which the body is brought from its rest to a position that is suitable for executing the gesture (cf. Figure 2.9a). The stroke phase contains the main part of the gesture (cf. Figure 2.9b), while in the retraction phase the body is brought back to its rest (cf. Figure 2.9c). In terms of gesture recognition, the stroke phase is the most important part of a gesture as it contains the actual information.

Other human computer interaction scientist as well used parts of the taxonomies by the social scientist or at least categories that describe similar properties. For example, Wobbrock et al. [189] used the categories



(a) Preparation (b) Stroke (c) Retraction

Figure 2.9: McNeill's gesture phases

symbolic, physical, metaphorical and abstract in the nature dimension of their taxonomy which are similar to Efron or McNeill, and Salem et al. [155] used the categories deictic, iconic, and miming, which is again close to McNeill. However, while the taxonomies of the social sciences are a useful tool to classify the style of a gesture, other properties of a gesture can be more important for the actual gesture recognition. Therefore, Wobbrock et al. [189] further defined the dimensions form, binding, and flow, apart from the nature dimension. In this way they could describe whether a surface gestures contained movement or not, how many touch points were present, whether it was related to an object or the whole touch screen, or whether the system already had to respond during the gesture performance. As a result of the different nature of full body gestures, there are again different properties more important for the actual gesture recognition. However, there does not yet exist a standard taxonomy for this kind of gestures. Therefore, I describe an own taxonomy in the Section 4.1.

Other researchers of computer science as well as the social sciences developed various coding schemes for describing human gestures and postures in more detail. An early example was the posture scoring system by Bull [21]. It defined postures in relation to a neutral pose, in which a person was sitting straight on a chair. For the different body parts head, trunk, arms, hands, legs, and feet, a posture was defined by how much the body part

was different to the neutral position, e.g. the head was raised. In addition, Bull defined a posture by whether one body part was touching another, e.g. a hand was touching the head. Kipp et al. [91] focused on hand positions, but defined them in a more comprehensive way, using the five dimensions height, distance, radial orientation, arm swivel, and hand-to-hand distance. Tan [172] combined the two previously mentioned schemes in her expressive postures (EXPO) coding scheme. The EXPO scheme used the dimensions height, distance, radial orientation and swivel as Kipp et al. and targeted full body postures as Bull. However, Tan included more body parts, e.g. the shoulders, and distinguished between sitting and standing postures. She further related various postures to action tendencies as introduced by Frijda [52, 53]. Unfortunately, Tan only presented a small number of samples, and those already included multiple ambiguities. This resulted in very low recognition rates in a study in which participants were asked to assign action tendencies to presented pictures of body postures. Dael et al. presented the body action and posture coding system (BAP) [33], which was at first very similar to Bull's system. However, they additionally distinguished between the actual pose and the movement towards that pose and add some more specific postures, e.g. "one arm holds other in front". Furthermore, they added eye gaze, (repeated) linear movements of specific joints in basic directions (left, right, upwards, downwards, forwards, backwards), and gestures according to the categories emblem, illustrator, beat, deictic, and manipulator, which was a mixture between the categories of McNeill [121] and Ekman and Friesen [44]. The presented coding schemes were already more helpful regarding gesture recognition technology, as they described the gestures in a more concrete way. Therefore, it is no surprise, that Velloso et al. [179] presented the AutoBAP system, which was a recognition system that automatically annotates gestures and postures according to the BAP coding system. They therefore made heavily use of worn tracking devices by employing a motion tracking suit in combination with a mobile eye tracker. This allowed them to cover a large amount of the BAP system, however, in many application scenarios, it would be impossible or at least undesirable to require the user wearing that much artificial gear. The perception markup

language (PML) by Scherer et al. [157] was an approach that was inspired by the behavioral markup languages for virtual agents. PML defined three layers for describing perceived non-verbal behavior: sensing, behavior and function. The sensing layer consisted of more or less raw sensor data, e.g. joint positions, orientations, gaze direction, but also postures or gestures. The behavior and function layers were meant to interpret the sensor data on a higher level. For example, the sensor layer could display an averted head orientation and the behavior layer interpreted this with a low attention score. While the idea of different layers for sensing and interpretation was generally a good idea, PML was not fully specified, yet, and the interpretation of the sensing layer probably would pose a bigger problem than defining a proper output format for it.

2.3 Posture and Gesture Recognition

For recognizing body gestures and postures, one needs to interpret the configurations and movements of certain body parts, i.e. joints. There exist different techniques to achieve that interpretation. In the classic gesture recognition problem, one has a vector of points (*here:* joint positions) as input and needs to compare it to different vectors of points that describe a number of gestures. By calculating which vector is most similar to the input vector, one classifies it to determine the gesture it might describe. In advanced recognition systems, the possible gesture points are represented by a model which is created manually or automatically out of multiple samples. However, in full body interaction the positional data as used in classic gesture recognition is only a part of the information you can get out the user tracking data as it represents the path of single joints, which mainly describes symbolic gestures usually performed with one hand, less often with two hands or other body parts. Apart from this, full body interaction needs to interpret the relation between different joints to recognize body postures, and the changes in those relations to recognize body gestures that have a less symbolic nature, e.g. dancing moves or clapping one's hands.

In the next section, I describe different approaches and algorithms to

achieve gesture recognition, ranging from the classic machine learning algorithms to recent approaches that target full body interaction in a more pragmatic way. Afterwards, I describe several frameworks that incorporate the gesture recognition approaches and provide further help to include full body interaction in applications.

2.3.1 Algorithms

A lot of work has already been done in the field of classic gesture recognition algorithms, and different approaches were developed and extensively tested, for which, e.g. the survey by Mitra and Acharya [125] gives a good overview. I will present popular approaches which seem to be promising for full body gestures in the following.

Statistical Classifiers

Statistical classifiers [151, 112, 163] regard the recognition task as a classification problem and try to solve it using machine learning and usually a supervised learning strategy for pattern recognition. Therefore, the algorithm first extracts a number of geometrical or algebraic features, e.g. the movement angle at the beginning of the gesture, the gesture path's bounding box size, the number of turning points, or the coordinate points themselves. A classifier is then trained with those features using a number of sample gestures for each possible gesture class. The training usually results in weights of how important each feature is for a certain class and how the features are combined mathematically to provide a score or probability that an input belongs to that class. During the actual application of the gesture recognition, the classifier tests an input gesture against each gesture class, using the same feature extraction and decides, on basis of the resulting score, to which gesture class the input may belong to. Different classifiers can be chosen including Naive Bayes, Decision Trees, Support Vector Machines, Neural Networks, and others. Some approaches further try to improve the overall accuracy by combining multiple classifiers, e.g. using a boosting algorithm [51].

Depending on the classifier, the robustness as well as the computational costs vary largely. Nevertheless, in case they are trained with enough data samples and well-chosen features, statistical classifiers are considered as very robust in recognition. Other advantages are that it is relatively easy to add new gestures by re-training and you can easily get a measure on whether there are any conflicting overlaps between the gestures. The feature extraction can usually be computed efficiently. The classification can as well be computed efficiently if a simple classifier is chosen [151], but the computational costs can increase dramatically with a more complex one. General disadvantages of statistical classifiers are that it can be quite difficult to choose good features for distinguishing between certain gestures, and that those features might need to be changed when adding new ones. They further rely on a large set of example data that they are trained with, although there have been improvements made to make them usable with smaller training sets [112]. Depending on the chosen features and training samples, the classifier can be very dependent on the rotation, size or speed of the gesture, although this may not always be wanted.

Finite State Machines

Finite States Machines (FSMs) are a theoretical model that can be used for gesture recognition [35, 16, 70, 146]. FSMs structure the recognition process into states $S = \{s_0..s_n\}$, which all correspond to a certain part of the gesture. An FSM always has a clear initial state s_0 and it can have one or more final states $F \subseteq S$. In addition to the states, FSMs have an input alphabet Σ and a transition function $\delta = \{\delta_{ij}$ with $i, j \in [1..n]\}$. As long as the gesture does not contain any repetitions, the state machine for recognizing the gesture is normally modeled linearly without any branches, so the process can either stay in the current state or move to the next one. However, all states can be aborted by going back to the initial state. The initial state itself usually represents that the gesture has not even started, yet. The transitions to the other states are taken according to δ as soon as certain input symbols of Σ occur. Again, the continuous input of a gesture cannot be used directly, instead rules are defined that interpret the current

input. Usually the rules define a certain range that the input values have to be in, e.g. the two-dimensional position of a stylus is in the lower left corner of the screen, or the movement direction is roughly going north which are the actual input symbols to the state machine. When comparing FSMs with statistical classifiers, these transition rules can be seen as simple decision trees. For example, the FSM of Figure 2.10 could model a "V" being drawn

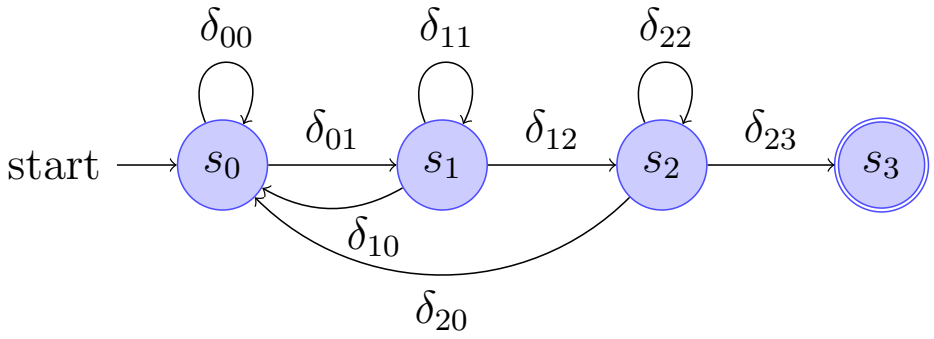


Figure 2.10: A linear left-to-right finite states machine (FSM) with four states and without branches, but with abort transitions

with a stylus on a touch screen. In the initial state s_0, the FSM waits for the gesture start. As soon as the stylus is in the upper left corner of the screen, which is tested by δ_{01} and δ_{11}, the gesture starts and causes the FSM to enter s_1. Similarly, s_2 is entered as soon as the stylus is in the bottom center (tested by δ_{12} and δ_{22}), and in s_3 the stylus needs to be in the upper right corner (tested by δ_{23}). All remaining transitions (δ_{00}, δ_{10} and δ_{20}) are taken in case, none of the other outgoing transition's rules are fulfilled. In addition, the states can have certain time constrains, e.g. a maximum duration [70]. In the FSM of Figure 2.10, the transitions δ_{10} and δ_{20} could be automatically taken after e.g. the current state lasted for 2 seconds already. This filters out gesture candidates that are not performed fluently and therefore are less likely to be actually meant as an input. A gesture is recognized as soon as the FSM reaches a final state (s_3 in Figure 2.10). Therefore, they have a clear decision on whether a gesture is recognized, in opposite to statistical classifiers and other techniques in which a score or probability indicates the recognition. The number of states and the conditions for the transitions can either be defined manually or they are

trained using sample data and a clustering technique. In the special case that there only exists an initial state and one final state, but no intermediate states, the FSM does not model progress any more, but only defines a set of rules which define a static gesture, i.e. a posture.

Advantages of FSMs are that they are easy to implement and the recognition is usually very efficient. They also model the gesture in an intuitive way, so that the transition function often can be defined manually instead of training it with sample data. The related disadvantage is that FSMs are usually very sensitive against even brief inaccuracies. An improperly defined transition can prematurely abort the whole recognition process.

Hidden Markov Models

Hidden Markov Models (HMMs) are another automaton based model which can be used for gesture recognition [194, 138, 129]. In opposite to FSMs, HMMs add a stochastic component, and the states do not clearly represent parts of gestures, but are invisible (hidden) to the observer. HMMs consist of a set of states $S = \{s_1..s_n\}$, a set of observations (= alphabet or set of input symbols) $Y = \{y_1..y_m\}$, the initial distribution $\Pi = \{\pi_1..\pi_n\}$, transition probabilities between those states $A = \{a_{ij}$ with $i, j \in [1..n]\}$, and the emission probabilities for each state that describe how likely it is to observe a certain symbol at a given state $B = \{b_{ij}$ with $i \in [1..n]$ and $j \in [1..m]\}$. HMMs are used for gesture recognition in a relatively similar way as statistical classifiers. They as well define several features that the HMMs are trained with to prepare for the actual recognition process and the same features are extracted from input gestures that should be investigated. The chosen features describe the set of possible observations, i.e. the input symbols. A default HMM has discrete input symbols, therefore a continuous input gesture first needs to be quantized or encoded in some way. For example, considering a two-dimensional gesture that can happen within a 1.0×1.0 space, this space can be split into 100 squares of size 0.1×0.1 indexed from 0 to 99. A gesture is now represented by a sequence of square indexes. Similar, a gesture can be represented by a sequence of directions, i.e. angles which are again clustered into certain ranges e.g. every

5° from 0° to 360°. To get less dependent on the orientation of the gesture, one can as well use direction changes clustered from −180° to 180° instead of actual directions. After this encoding step, the discrete symbols are used as input for the HMMs. There also exist continuous HMMs that directly use continuous input and therefore, the encoding step is unnecessary. Nevertheless, many approaches still use discrete HMMs as they are easier to implement. HMMs for gesture recognition usually have a fixed start state ($\pi_1 = 1$), a fixed end state (s_n has no outgoing edges) and a specific number of states in between. The number of states is usually overspecified,

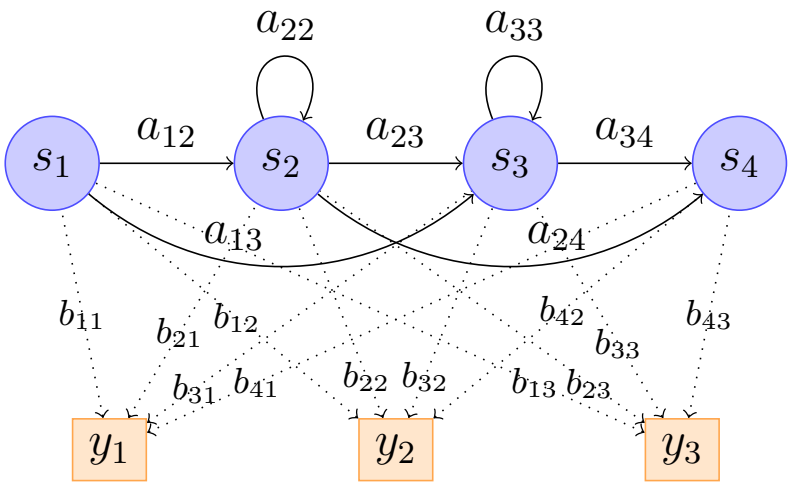


Figure 2.11: A linear left-to-right hidden Markov model (HMM) with four states, three emissions and the possibility to skip states

e.g. when a gesture only consists of two distinguishable parts that could be modeled in states as drawing a "V", the HMM is specified with four states as shown in Figure 2.11. In this way, the HMM usually gets more robust against slight inaccuracies in the gesture performance. The topology of the HMMs is often strictly linear from left to right with [150] or without [113] the possibility to skip a state (cf. Figure 2.11), sometimes with a cycle from the last to the first state if the gesture contains repetitions [150]. However, other researchers also use more complicated topologies with up to transitions between all states [3]. Using multiple samples for each gesture class, an HMM is trained for that specific gesture, e.g. using the Baum-Welch algorithm. The training determines the transition and emission probabilities,

which now completely defines the HMM. The actual gesture recognition compares the input gesture with different gesture classes, e.g. by applying the Viterbi algorithm on each of the corresponding HMMs.

The advantages of HMMs are similar to the statistical classifiers, as they are regarded as robust and efficient in recognition as well. In the same manner, they again rely on adequate features and training. Another difficulty with HMMs is the selection of an appropriate topology, especially as the most intuitive ones usually do not yield the best results [3].

Dynamic Time Warping

Dynamic time warping (DTW) [128] is a dynamic programming technique that was originally developed to achieve speech recognition at different speech rates, but soon proved to be useful for gesture recognition with different gesturing speeds as well [176, 31, 191]. Since the early beginnings, several improvements have been developed to increase its performance or accuracy [2]. DTW tries to compare temporal feature sequences while allowing them to have different speeds in parts of the sequences. For this purpose, DTW calculates a warping path that determines the optimal way to match the elements of one sequence to the elements of the other sequence, while keeping their temporal order. For example, in Figure 2.12, two one-

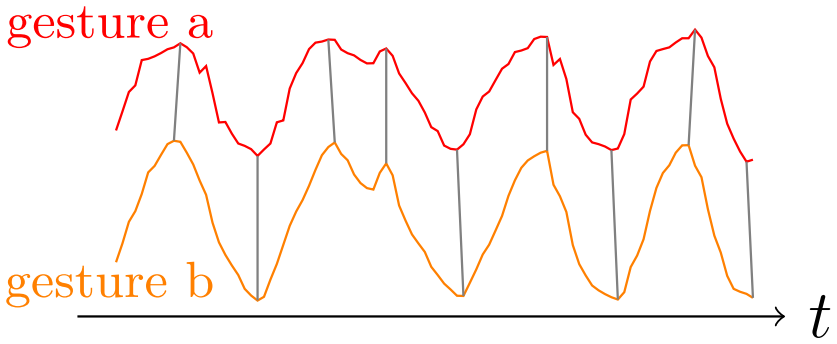


Figure 2.12: Dynamic time warping (DTW) applied on two 1D gesture paths a and b; gray lines visualize parts of the warping path

dimensional gesture paths are defined by the red and orange curve. Parts of the warping path are visualized by the gray lines that match elements

of the two gesture paths. In parallel, DTW calculates the costs for matching the sequences, i.e. their distance in sense of the DTW. This distance consists of the sum of distances between the matched sequence elements according to a certain distance measure. For gesture recognition, it is usually enough to have a single template for each gesture class, which can be a recorded gesture performance or a manually defined optimal gesture path. During the recognition, the DTW distances between the input gesture and the templates of all gestures classes are computed. The distance between two data points of the gesture is usually calculated by the Euclidean or Manhattan distance. While the Euclidean distance is the default way to calculate distances, the Manhattan distance only "allows" movements along the coordinate axis and therefore rates distances higher if they involve multiple axis. Kristensson et al. further achieved better results [103, 104] using the angle between the movement directions of the data points as a distance measure, here called "turning angles". When the average distance between all pairs of data points is below a certain threshold, the gesture class is considered a match. In case multiple classes stay below the threshold, the class with the minimal distance is taken.

Advantages of DTW are that it is relatively robust against slight variations without requiring extensive training. Further, it explicitly allows different gesturing speeds, and therefore, a resampling step as done in the later described $1 recognizer is unnecessary. Nevertheless, other normalization steps similar to the rest of the $1 recognizer can be helpful. Disadvantages are that DTW is relatively computational expensive, especially if there are many and/or long gesture classes.

Dollar $1 Recognizer

As most of the aforementioned recognition methods are quite difficult to apply for practitioners, e.g. because they require complex implementations, well-chosen features, or large training data, Wobbrock et al. [190] developed the $1 recognizer which should make gesture recognition more accessible to developers without a strong background in pattern recognition and since then, different improvements and variants have been published under the

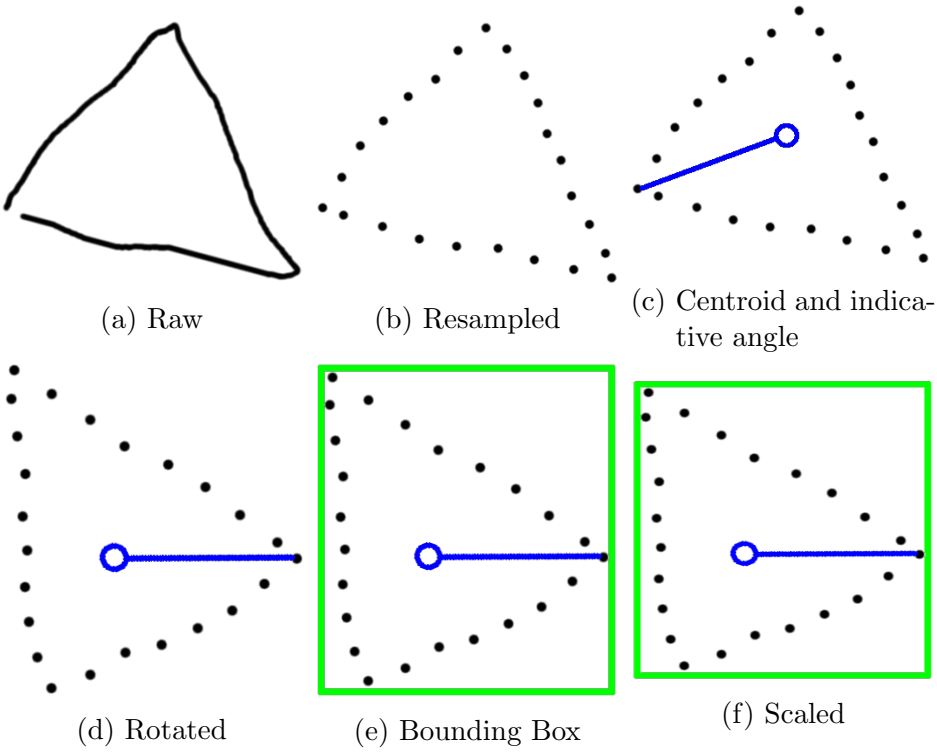


Figure 2.13: Normalization steps of the $1 recognizer

"$-Familiy" [111, 5]. The $1 recognizer incorporates a simple approach for comparing gesture paths, i.e. a K-Nearest Neighbours (KNN) technique, but uses multiple normalization steps that make the recognition surprisingly robust. The normalization steps are depicted in Figure 2.13. First, the raw input (cf. Figure 2.13a) is resampled so that the gesture path is represented by a fixed number of equidistant points, e.g. 24 points as in Figure 2.13b. The resampling goes through the raw input, skips points if they are closer than the desired increment to the last added point, but if they are further away, it interpolates linearly between the last added point and the current one to be exactly one increment away and adds the interpolated point to the output path. Afterwards, the gesture path is rotated around its centroid so that the indicative angle (= angle from centroid to first point of the path) is zero which is usually defined by pointing in the direction of the positive x axis (cf. Figure 2.13d). Now, the path is scaled non-uniformly to transform its bounding box to a reference square and it is translated to have the centroid at the origin (cf. Figure 2.13f). Those normalization

steps are applied to the sample gesture of the gesture classes as well as to an input gesture performed during runtime. The actual comparison between the input gesture and the sample gesture of a gesture class is done by calculating the average pair-wise point distance. To further improve the results, Wobbrock et al. use a Golden Section Search strategy to find the optimal rotation angle for aligning the two gestures around ±45° from the indicative angle within ten iterations.

The main advantage of the $1 recognizer is that it achieves robust recognition with a simple and efficient algorithm that even outperforms traditional recognition techniques as the Rubine statistical classifiers [151]. The disadvantages are that, although the normalization steps make the recognition robust against variations of the gesture, they also prevent that the $1 can distinguish between gestures that require certain orientations, sizes, aspect ratios, locations or movement speeds. For example it cannot distinguish squares from rectangles, left-arrows from right-arrows, and does not work with straight horizontal or vertical lines at all. Those shortcomings can be overcome by modifying the algorithm and omitting unwanted normalization steps. Nevertheless, omitting too many steps can make the recognition less robust again.

Another interesting recognizer which can be seen as an enhancement to the $1 is the poly line recognizer by Fuccella and Costagliola [54]. They change the resampling in a way to achieve a gesture consisting of as few as possible connected lines. In the recognition process, they then apply an alignment which again adds points to the template or the input gesture from their original gesture path in an intelligent way to get an equal number of points for the actual recognition. Fuccella and Costagliola claim to achieve clearly better recognition results with this approach in comparison to the $1 (or more precisely its successor the Protractor [111]).

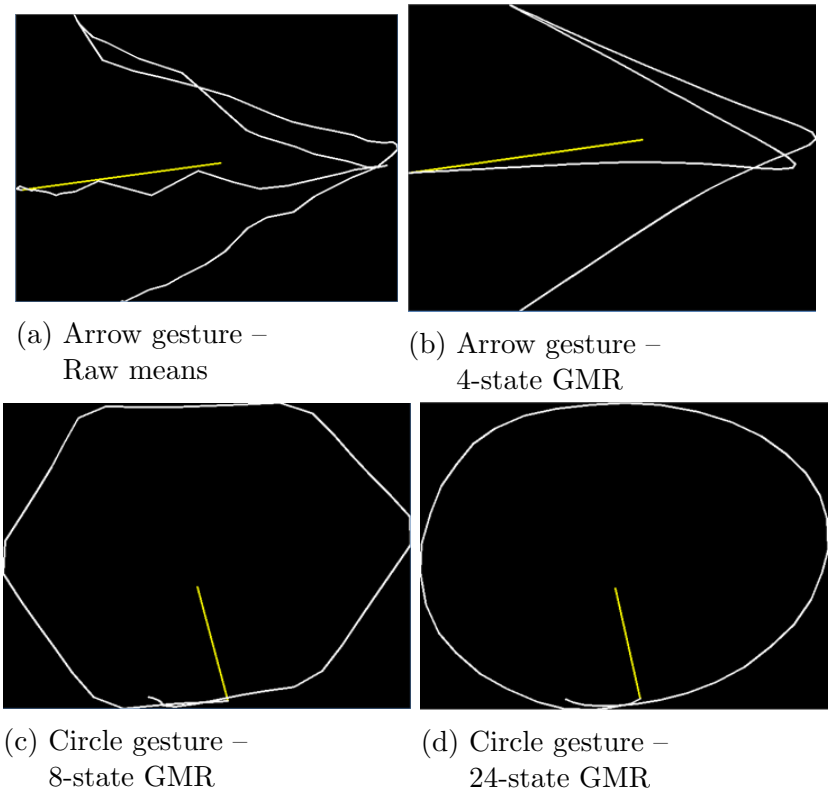
Stochastic Modeling and Gaussian Mixture Regression

Some of the basic gesture recognition techniques, e.g. $1 and DTW define a gesture class by a single template. Therefore, those techniques do not necessarily represent a generalized and user-independent form of the

gesture and they may not recognize all wanted geometrical variations of the gesture. Nevertheless, this shortcoming can be overcome by adding a statistical model which is calculated out of multiple gesture samples.

Basically, one can calculate **means and covariance matrices** for the data points of the gesture samples and use those to calculate the **Mahalanobis distance** to an input value instead of the Euclidean or Manhattan distance. The Mahalanobis distance measures the distance in relation to the provided covariances and therefore values a distance less if there also is a high variation in the corresponding part of the training samples, but values the distance more if there is only few variation in the training samples.

To further improve the jerky gesture paths generated with only few gesture samples, another option is to calculate a **Gaussian mixture model (GMM) and** apply **regression (GMR)** to achieve a much smoother ges-



(a) Arrow gesture – Raw means

(b) Arrow gesture – 4-state GMR

(c) Circle gesture – 8-state GMR

(d) Circle gesture – 24-state GMR

Figure 2.14: Calculating a mean gesture path of four samples using Gaussian Mixture Regression (GMR)

ture path (cf. [23]). The GMM separates the gesture in multiple states for which the means and covariances are calculated using an expectation maximization algorithm. Using regression, new coordinates and covariances are again calculated for each point in time. For example, Figure 2.14a displays the mean coordinate values calculated out of four samples of an arrow gesture, while Figure 2.14b displays the new coordinates calculated out of the same four samples of an arrow gesture, but by first computing a GMM with four states of equal length and then applying regression. Nevertheless, using this method, one has to take care of choosing a good number of states for the GMM, e.g. in Figure 2.14c only eight states are used for a circle gesture resulting in a very angular path, while in Figure 2.14d, 24 states are used to achieve a much rounder one.

HMMs and statistical classifiers already include a statistical model and therefore do not profit from such techniques. The rules for the states of an FSM can however be learned from multiple gesture samples as well (cf. [16]). In this way, FSMs can also be enhanced with statistical modeling.

2.3.2 3D Space and Temporal Data Segmentation

Most of the aforementioned recognition approaches were originally targeted at gestures defined by a temporally presegmented movement of a single point in 2D space. Start and end were indicated to the algorithm, either in an implicit way, e.g. by touching the surface of a PDA with a pen [190], or in an explicit way, e.g. by pressing a button on a Wiimote [101]. Therefore, there are several challenges when porting the recognition algorithms to unsegmented multipoint 3D data as in full body interaction.

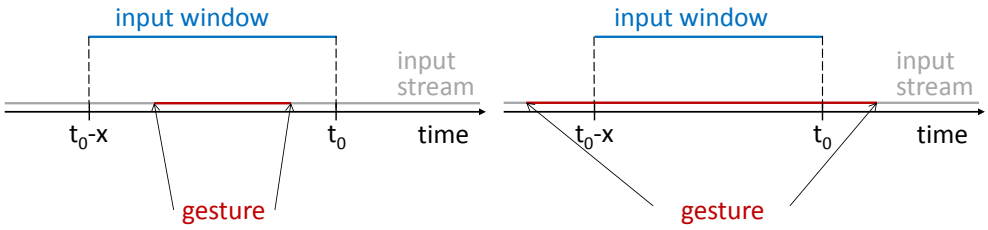
Porting the recognizers to 3D space usually only makes the algorithms mathematically more complex, but is feasible in general, e.g. Kratz and Rohs ported the $1 recognizer to 3D space [101].

Temporal data segmentation is generally not important for FSMs. They can work with continuous data streams and update their status whenever new data arrives. When using appropriate algorithms, the same applies for HMMs [38]. Other techniques, such as statistical classifiers, are more dependent on temporal data segmentation, as they are meant to compare com-

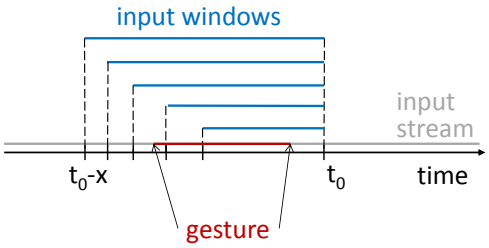
plete input gestures with a templates. Therefore, those techniques require some kind of presegmentation or at least an incremental update mechanism.

Some approaches for device-free interaction actually apply manual data segmentation. For example, one can use an **accompanying posture**, e.g. raising one hand, while performing a gesture with the other hand [97]. The temporal segmentation is given by the start and end of the accompanying posture, i.e. it is assumed that the to-be-classfied input gesture lasts exactly as long as the easy to detect posture. A similar approach has been presented by Gu et al. [60], who use a **starting pose** for indicating the beginning of a gesture, e.g. raising the hand for the gesture waving. Here, only the start of the gesture is determined by the pose, while the end of the gesture is defined by a fixed duration, i.e. 1.5 seconds. An alternative would be keeping a **minimum distance** of the hand to the body while gesturing [97]. Technically, this is still an accompanying posture, however, it is performed with the gesturing hand itself. The gesture start is assumed as soon as the hand gets further away from the body than the minimum distance. As soon as the hand gets again closer to the body as the minimum distance, the gesture is ended. This is similar to the **zoning** technique by Kristensson et al. [104], who further require that the speed of the gesturing hand and the body as a whole stay below a certain threshold, i.e. the gesture won't start or will be stopped in case there are too fast movements. Ni et al. [132] use special hand postures, called **delimiters**, to mark the start and end of a gesture. They therefore use a colored glove for recognizing the correct hand posture robustly. All of those solutions are targeted at one-hand gestures, but would not work well for other full body gesture types in which more body parts are involved. In general, they make the interaction more complicated, unnatural, and still do not ensure a precise segmentation, as it can be quite difficult for the user to indicate the gesture duration.

There are also automated techniques to tackle the missing temporal segmentation. A brute-force segmentation approach applies the recognizer in every frame on a **sliding window** of buffered data as illustrated in Figure 2.15, e.g. applied by Peng et al. [142]. This is computationally much more expensive than applying the recognizer as soon as a presegmented



(a) Oversized sliding window (b) Undersized sliding window



(c) Multiple parallel sliding windows with different sizes

Figure 2.15: Temporal data segmentation with sliding windows

input arrives, but can be feasible for real-time recognition in case of a small gesture set or very efficient recognition algorithms. A sliding window also dramatically increases the probability of false positive recognitions, as much more data is analyzed that actually does not represent any gesture at all. Furthermore, depending on the window size, the gesture performance may not be covered completely (cf. Figure 2.15b) or a lot of additional data will remain around the gesture (cf. Figure 2.15a). Both of those cases can make the recognition impossible for certain techniques, especially for statistical classifiers and the $1. For dynamic programming techniques such as DTW, the recognition may still be possible, but it can get less accurate and a certain delay may be induced. One solution can be, to analyze data windows of different sizes to find the best match to a gesture template (cf. Figure 2.15c), however, this further increases the computational complexity and the probability of false positives. Especially for statistical classifiers or HMMs, it can as well be helpful to have an own model for motion data that does not represent any gesture, i.e. non-gesture or garbage classes [138]. As the garbage classes should more or less cover all motions except for the

actual gesture set, it can be hard to define or train those classes.

Therefore, other techniques try to segment the data stream before applying the actual recognition, e.g. Zinnen and Schiele [201] use **turning points** of arm movements to find segments of interest within the data stream that a statistical classifier is applied on. However, this approach only works for gestures that have enough distinctive turning points in their path. Other researchers use the hand speed and acceleration to segment hand gestures. For example, Bhuyan et. al [13] try to segment hand gestures in the phases pause, preparation, stroke and retraction adopted from McNeill [122]. They therefore use fuzzy logic rules regarding the speed and acceleration of the hand. Zhu and Sheng [200] use a neural network for this task. Again, these approaches only apply for limited scenarios. Other researchers divide the gesture classes into **sub gestures** for improving the recognition without given segmentation. For example, Kristensson and Denby [103] regard multiple stages of progress for a gesture class. The corresponding sub gestures are first evaluated separately, and the whole gesture class then receives the best score of its sub gestures, while the score of the sub gesture with complete progress (the whole gesture) is weighted more important. However, the algorithm by Kristensson and Denby still requires a partial segmentation, i.e. the start of the gesture should be known, and it is still computationally expensive. Other approaches intertwine segmentation and recognition by combining several recognition methods, e.g. Alon et al. [2] combine a Gaussian model with DTW and a sub gesture model to achieve a good compromise in the recognition of video data without full spatiotemporal segmentation of hand gestures.

2.3.3 Frameworks

The release of the Kinect again motivated researchers to investigate gesture recognition for the more complex motion capturing data provided by full body tracking. The challenge with this data is that it contains multi-point data in 3D space (position and orientation of multiple important joints of one or more users). In addition, there is no obvious way to apply manual data segmentation (no device in the users' hands to press a button on),

and the data itself also is rather noisy [89]. On the other hand, this means that the data already contains more information as in other interaction modalities. In this way, one single data frame for one tracked user can already be seen as a gesture, or more precisely a posture, as it defines a specific configuration of the user's skeleton. For these reasons, researchers have developed easy to use rule-based techniques which can be seen as a basic FSM with an initial and a single final state as mentioned in the previous section. With those techniques, it is possible to achieve full body gesture recognition without any manual data segmentation, but by using the richer information of full body tracking.

One of the first frameworks for gesture recognition with the Kinect that uses such an approach, has been presented with the **Flexible Action and Articulated Skeleton Toolkit (FAAST)** [169]. It defines gestures with simple text scripts that contained restrictions for specific joint position in relation to other joints. It initially bound gestures defined by simple text scripts to mouse and keyboard events to control arbitrary applications via full body interaction. This was a very pragmatic and popular approach, and soon, a large number of non-technical users created gesture scripts for many different popular computer games to show that they can be controlled via Kinect. However, the "Kinectified" games were originally designed for traditional input such as mouse and keyboard and therefore not well suited for Kinect input. In addition, the gestures itself were quite limited (at first FAAST only supported simple postures as "left hand more than 20 centimeters left of the body"). Therefore, the enthusiasm about FAAST stopped at some point. In a later version [170], the authors made their framework more usable by adding a GUI for editing the gesture scripts and they further increased the recognition capacities by allowing to concatenate multiple gestures to more complex sequences. One remaining problem with FAAST is that it does not know about the state of the listening application and therefore cannot reduce the current gesture set to the gestures actually meaningful to the application at a certain point in time. FAAST is freely available as a pre-compiled binary, but no source code is published.

The **ProximityToolkit** [118] gathers proxemics information about peo-

ple and objects in the scene, e.g. position, orientation, current movement speed, direction and relations between different entities such as the distance or whether a user is facing an object. It can be seen as a full body interaction framework as it uses full body tracking data and the provided proxemics information is a special kind of postural information. However, it does not provide actual gesture recognition in the way the other mentioned frameworks do.

Kinetic Space [191] uses a different approach to achieve easily applicable gesture recognition with the Kinect. It provides a graphical interface to record gestures and uses the recordings to apply gesture recognition on the live data stream using the dynamic programming technique dynamic time warping (DTW) [128]. Therefore, it uses a brute force windowing approach, i.e. it applies the gesture recognition in each frame by passing the data of the last x frames, with x being a fixed window size. This allows more complex gesture shapes, but it is computationally more expensive, introduces a certain delay and is not well-suited for shorter gestures. Kinetic Space [191] had initially been developed as an open source framework for applying DTW with the Kinect. It was later commercialized and changed to closed source. Kinetic Space is a GUI application with two main modes: training and recognition. In training mode, gestures are recorded by performing them in front of the sensor, and afterwards weighting the joints according to how important they are for the current gesture and selecting the frame range that represents the gesture in the recorded stream. In recognition mode, all trained gestures are recognized online and recognition events are distributed using the OSC protocol. The advantages of Kinetic Space are that it requires no programming skills and allows to easily integrate quite complex gestures as the user only needs to control the provided GUI-interface and to perform the template gestures. Disadvantages are that the user gets no information why a gesture was not recognized and he or she has also no option for tuning a gesture after recording. Similar to FAAST, Kinetic Space has no information about the current application state and therefore needs to observe all available gestures at any time.

Full body tracking and gesture authoring functionalities are provided

in the **Omek Beckon Framework** (http://www.omekinteractive.com (accessed 2014-3-10)). Gesture authoring happens similar to Kinetic Space by recording a gesture performance and later recognizing the gesture using non-specified machine learning techniques. As the Omek framework has been discontinued after the acquisition by Intel and it is no longer available, it is mentioned here for the sake of completeness only.

The commercial gesture recording and recognition toolkit **GesturePak** (http://www.franklins.net/gesturepak.aspx (accessed 2015-9-15)) is a tool for integrating Kinect interaction in .NET 4.0 applications. Users can record gestures that are defined by a sequence of poses. For that purpose, they have to perform the different poses and speak out a keyword for recording it in an XML file. Before the recording, they can further select the joints that should be recorded. During run time the application compares the recorded gestures with the current poses of the users and reports any matches. GesturePak is closed source and the website does not provide information about how the gestures are recorded or compared. However, the recording process itself only allows very limited options.

The **Generic Multi-Modal Natural Interface Framework (GeMiNI)** [177] is a framework for integrating multiple interaction devices into arbitrary applications by generating mouse and keyboard events. Among others, they support the Kinect, for which pose recognition has been implemented. Poses are defined by one or more spatial relations between two joints, e.g. distance, in front, above, etc. In addition to editing those values manually, the GeMiNI framework also allows to record poses in a short training session of about five seconds. As the poses are defined in absolute coordinates, they are not rotation invariant, but rely on that the user is standing with a specific orientation and position to the sensor. GeMiNI is a research project that is unfortunately not yet available for download.

ProtUbique [87] is a framework for prototyping interactive ubiquitous systems, i.e. systems that unobtrusivly surround the user with computing devices and support the user with information as well as interfaces. Among other functionality, ProtUbique supports posture and gesture recognition with the Kinect. This is regarded as a suitable interaction modality as

the user does not have to pickup a handheld device or special gear and the interaction device is therefore unobtrusive in this sense. Postures are defined in a graphical interface that displays a 2D skeleton consisting of the available user joints visualized as circles and connected by lines. Two body joints can be arranged in relation to a third "base joint" by drag-and-drop to define a posture. Gestures can further be defined as sequences of postures. Although the description of this functionality is very brief, it can be assumed that the gestures defined in this way are very limited (only two-dimensional and no further control over the timing). However, the authors also suggest that gesture recognition might be handled by third party components in a future version of ProtUbique. ProtUbique is a research project that is unfortunately not yet available for download.

GestIT [165] is a framework for integrating full body as well as multi-touch gestures. Both gesture types are defined in a declarative way, however, ground terms have to be defined in native code, e.g. C#. Ground terms are restrictions for the positions or orientations of tracked features, e.g. "right shoulder is in front of left shoulder" or "right hand moves right". Those ground terms can further be composed with operators for defining temporal relationships providing more gesture varieties. Those operators are defined in XML. As the ground terms have to be defined in native code, GestIT requires coding skills as well as deep understanding of the tracking data. The composition of ground terms would pose a convenient way to define more complex gestures and offers many ways to define synchronizations of gestures, e.g. iterations, parallelism, choice, any-order sequences. However, the XML language is quite complicated and does not provide time-based constrains, e.g. minimum or maximum duration. GestIT is published as open source, but without pre-compiled binaries.

The **Customized Body Gesture Design (CUBOD)** tool [173] is a tool that lets end users design gestures by recording their gesture performance in a GUI and uses DTW for gesture recognition, similar to Kinetic Space and with the same advantages and disadvantages. What distinguishes CUBOD from Kinetic Space and other frameworks is the feedback, users get for a created gesture. CUBOD provides information on how consistent

multiple sample performances of a gesture are, whether there are other gesture classes that could be too similar to the proposed gesture, and whether a gesture is not distinctive enough from unintentional movements. In this way, users get a measure on the quality of their proposed gesture and can revise it. However, the authors admit that their focus lay on the gesture design and not on the recognition and the tool is not published, yet.

GDL [62] is an abbreviation for gesture description language, and allows to define full body gestures according to one or more rules in a custom scripting language. In the scripting language, one has access to the positional data of fifteen tracked joints (i.e. the OpenNI 1.x skeleton) with a history of up to 5 seconds. Furthermore, several mathematical operations can be applied on that data, e.g. basic operations such as +, −, but also more complex ones such as *sqrt*, *distance*, or *angle*. In addition, relational operators against other joint values or constant numbers can be used and further concatenated with boolean operators to finally formulate the rules. To support sequences, the operator "sequenceexists" further allows to test whether a sequence of rules happened in the last five seconds while conforming to specified time constraints. A test application for GDL is published freely in binary form, but without access to the source code or the possibility to use the recognition functionalites in an own application. At least the latter is promised to be possible on request on the website.

XKin [139] is an open source framework for recognizing hand postures and gestures using texture-based descriptors and HMMs with discrete angular features. The framework is meant to recognize the American Sign Language (ASL) alphabet and 16 additional uni-stroke gestures, but can be extended with user-defined gestures as well. The framework does not make use of user tracking, but works on the depth stream only. It only works with one-hand gestures and requires a specific setup in which the user is holding the hand in direction of the sensor within a certain distance.

XDKinect [130] is meant to facilitate development of cross-device applications with the Kinect in a client-server architecture. Initially, it was meant to allow access to Kinect tracking data from within a browser using JavaScript code. It provides access to speech, skeleton tracking and the

built-in hand gestures of the Kinect SDK 1.x as well as proxemic features. Additional gestures can only be added by analyzing the tracking data using own recognition techniques within the scripting language. XDKinect is a research project that is unfortunately not yet available for download.

EasyGR [74] is a framework for easily training and recognizing Kinect gestures within a C#-based game engine using DTW or HMMs with discrete input symbols achieved through k-means clustering. Ibanez et al. therefore implemented a simple graphical interface within the Unity3D game engine[10]. The interface allows to record and save training data to text files that can later be loaded for the actual recognition. EasyGR is a research project that is unfortunately not yet available for download.

Hotspotizer [11] is developed along a user-centered design approach. Users can declarative author gestures in a graphical interface by selecting cubic cells (hotspots) around the humanoid figure. By concatenating those hotspots to sequences, gesture paths can be defined. During the recognition, the different hotspots have to be visited in the given order with a maximum of 500 ms in between. Hotspotizer is implemented in C# and .NET, and therefore Windows specific. It is freely available for download, but no source code is accessible.

In the **FUBI** framework, I aim to achieve more powerful gesture recognition with various gesture types and complexer configuration options in an XML-based definition language. FUBI allows to define static postures and dynamic gestures that are recognized in a per-frame basis as well as incorporating FSMs to construct longer sequences. Furthermore it also provides a template based recognition technique which can be incorporate with other techniques such as dynamic time warping, gaussian mixture regression or HMMs. This allows integrating more complex symbolic gestures if needed. More details can be found in Section 5.4.

Apart from the third-party gesture recognition frameworks, it has to be mentioned that the tracking frameworks that target low-cost depth sensors, i.e. NiTE and the Kinect SDK, both offer basic gesture recognition functionality already. **NiTE v. 2.0** supports three simple one one-hand

[10] http://unity3d.com (accessed 2015-09-16)

gestures “wave”, “click”, and “hand raise” and two body postures “psi pose” and “crossed hands”. Especially the recognition of the hand gestures is not very reliable, the gestures cannot be modified at all and there is also no possibility to add new gestures. The **Kinect SDK v. 1.8** supports two very simple one-hand gestures available in its C#-API only: “grip” and “press”. Again, the gestures cannot be modified at all and there is also no possibility to add new gestures. This did not change significantly in the **Kinect SDK v 2.0**, in which three simple one-hand gestures are supported (this time at least in the main API): “open”, “closed”, and “lasso”. However, a more recent version of the Kinect SDK v 2.0, added a tool called Visual Gesture Builder which applies statistical classifiers in combination with the AdaBoost technique [51] for training gesture recognition based on several gesture performances. The user has to record several gesture samples and mark positive an negative gesture occurrences on a per-frame basis. Afterwards, the tool trains the classifier, first using a large number of features, but reducing them later with the AdaBoost technique. The tool allows to test the classifier in a live session or using recorded sample data, however, there is currently no option to include the gesture in an own application as the software is still in an alpha stage. Further, it seems that the classifiers completely works on a per-frame basis and therefore can only recognize static postures or simple movements. Therefore, the tool is practically not usable at the current stage. Overall, the gesture support by the common tracking frameworks is very restricted, and separate gesture recognition software is definitely needed.

Furthermore, there also exist multiple general frameworks and toolkits for machine learning, which directly target gesture recognition or at least can be employed for implementing it, e.g. **WEKA** [64], **Java-ML** [1], **SHOGUN** [164], **Scikit-learn** [140], or **GRT** [56]. They provide a huge variety of machine learning algorithms with APIs for different programming languages and they are designed to operate on large data sets. Nevertheless, none of those frameworks does directly target full body interaction. They do not integrate depth sensors and full body tracking, and they do not focus on the specific features of full body tracking data. Often, they even

do not target the real-time application of their algorithms, e.g. real-time gesture recognition in an online system, but they are mostly suited for offline evaluations. Therefore, I will not further investigate general machine-learning frameworks here.

Table 2.3 summarizes important features of the above mentioned full body gesture recognition frameworks. An "✓" in the column with the following labels (in italic) means that the corresponding framework supports the features as described here:

1. *finger gestures*: hand gestures that involve finger poses or movements
2. *body gestures*: gestures that involve multiple body parts (e.g. not only the hands)
3. *motions*: dynamic gestures that involve movement of a body part (e.g. not only static postures)
4. *sequences*: concatenation of basic gestures and/or postures to form longer sequences
5. *complex paths*: gestures defined by complex paths usually given via templates or samples
6. *predefined*: predefined gesture definitions that can be used instantly
7. *user-defined*: adding new gestures
8. *trained*: training gestures with sample performances
9. *multi-lingual*: API for multiple programming languages or streaming of data via sockets or key commands
10. *freely available*: availability and usage without any charge
11. *open source*: source code can be studied and used by anyone

Table 2.3: Comparison of full body gesture recognition frameworks

	finger gestures	body gestures	motions	sequences	complex paths	predefined	user-defined	trained	multi-lingual	freely available	open source
NiTE[1]			✓			✓			✓		
Kinect SDK	✓		✓			✓			✓	✓	
– Gesture Builder[2]		✓	✓		✓		✓	✓			
FAAST [170]		✓		✓			✓		✓	✓	
Proximity Toolkit [118]		✓				✓				✓	✓
GesturePak [3]	✓		✓		✓	✓	✓				
Kinetic Space [191]		✓	✓		✓	✓	✓	✓	✓	✓	
GeMiNI [177]		✓					✓	✓	✓		
ProtUbique [87]		✓					✓		✓		
GestIT [165]		✓	✓	✓		✓	✓		✓	✓	✓
CUBOD [173]		✓	✓		✓		✓	✓			
GDL [62]		✓	✓	✓		✓	✓				
XKin [139]	✓		✓		✓	✓	✓	✓	✓	✓	✓
XDKinect [130]	✓	✓									
EasyGR [74]		✓	✓		✓	✓	✓	✓			
Hotspotizer [11]		✓	✓	✓	✓		✓			✓	
FUBI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

[1] Official download site discontinued and therefore marked as "not freely available"

[2] Alpha version, currently not usable in an own application and therefore marked as "not freely available"

[3] Published on http://www.franklins.net/gesturepak.aspx (accessed 2015-9-15)

2.4 Freehand GUI Interaction

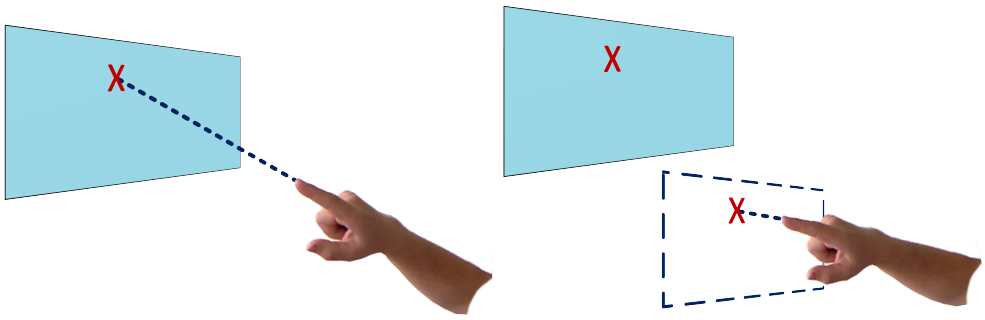
Posture and gesture recognition, as presented in the last section, are mainly used for a discrete (event-based) interaction. The user performs a certain gesture, the system recognizes it, and triggers the system action related to the gesture. Nevertheless, the input data of full body interaction can as well be used in a continuous way. For example, the hand motion can be used

continuously to control a cursor in a GUI. All current operating systems are based on GUIs usually controlled by mouse/keyboard or finger touch. When porting them to full body interaction without handheld devices, there are several differences the interaction designer needs to be aware of. Although a discrete full body interaction should be preferred for interaction in virtual environments (cf. Section 3.1) and most application scenarios investigated in this dissertation apply discrete interaction, there are cases in which it is worth to implement continuous full body interaction and, e.g. to control a GUI similar as with the mouse and keyboard (cf. Section 5.3).

A basic GUI includes a pointer or cursor that the user controls, as well as graphical items that are displayed on the screen. The user can select an item by moving the pointer to it and confirming the selection which is known as the point-and-click paradigm. Depending on the interaction modality, the mapping of the user input to cursor movements and the item selection differ. With conventional mouse interaction, the two dimensional mouse movements are mapped to the screen according to the settings of the operating system. Confirmation of an item selection is realized by pressing a button on the mouse. With touch interaction, the cursor movement and item selection can happen within one step, when the finger or stylus touches the surface. For full body interaction without handheld devices, the cursor is usually controlled with one hand in the air (freehand interaction), however, there are various options how to realize the cursor mapping and item selection. I describe several such options in the following sections.

2.4.1 Cursor Control

An intuitive way to realize cursor movement, when interacting from a distance without handheld devices, is pointing at interface items with one hand. To give feedback on the pointing position, the cursor usually has a graphical representation on the screen. In opposite to the conventional arrow representation in mouse interaction, for freehand interaction it is common to display a hand icon on the screen. There are different ways of mapping the hand position to a screen position, e.g. Vogel and Balakrishnan [182] distinguish between absolute ray-casting and relative pointing, which



(a) Absolute ray-casting (b) Relative pointing

Figure 2.16: Absolute ray-casting (a) and relative pointing (b) on a screen (light blue colored rectangle)

are illustrated in Figure 2.16. For ray-casting, the mapping is defined by the point in which a ray extended from the hand in pointing direction intersects with the screen (cf. Figure 2.16a). Therefore, users directly point at the objects on screen which is potentially more intuitive. Relative pointing applies an indirect mapping, in which hand positions relative to the user's body are mapped to screen positions, without taking the actual placement and dimensions of the screen into account (cf. Figure 2.16b).

Therefore, an indirect mapping provides higher accuracy when standing at a farther distance or pointing onto a smaller screen, and allows for a more comfortable (lower) hand position when standing closer or pointing onto a larger screen. For example, Vogel and Balakrishnan [182] measured 22.5% error rate for ray-casting in comparison to 3.5% for an indirect mapping with the task of pointing at small targets (16 mm) from a distance of 4 m. The higher hand position of the ray-casting technique represents a common problem of midair interaction, sometimes referred to as the "gorilla-arm effect" [67]. Because of the corresponding arm fatigue, the interaction gets less precise over time, and the user has to take more breaks or preliminarily stop the interaction earlier.

For small targets distributed on a larger screen, researchers further developed approaches, in which the cursor does not target a single point on the screen, but a (possibly adaptive) activation area [59, 168]. In this way, users do not have to point at the target precisely, but it is allowed to have a certain displacement, as long as this does not favor another nearby target.

2.4.2 Item Selection

With a working freehand pointing mechanism, the user is able to hover a cursor over GUI items, however, we still need to find a way to determine when the pointing actually indicates the selection of an item which is called the Midas Touch problem in the literature. In the case of mouse interaction, a selection can be simply elicited by a mouse click. For other interaction techniques, such as gaze, touch or freehand interaction, the mouse click has to be replaced by alternatives. There exist several solutions in the literature that can also be used within freehand interaction.

An easy way for taking over the function of a traditional mouse click is an automatic selection after a certain dwell on an indicated item. However, adapting dwell time to a particular situation and an individual user is a great challenge. On the one hand, dwell time needs to be chosen long enough to avoid false alarms. On the other hand, it should be rather short in order not to slow down user interaction. For gaze based GUI interaction, dwell times between 0.3–1 seconds are typically chosen and sometimes also adapted for expert users [115]. For full body interaction, higher dwell times need to be taken, as the interaction is slower as well. For example, Microsoft provides a dwell based method for interaction in the Xbox Kinect GUI and dwell time is set to about 1.3 seconds (Xbox Firmware 2.0.14719.0). Because of this fixed duration, such a dwell-based approach has a clear limit in performance, which does not offer many possibilities to improve by training. Nevertheless, a dwell-based selection is considered to be easy-to-use for novice users and to offer a constant low error rate.

Another solution to the Midas touch problem is the definition of a specific selection area on the screen. This solution is commonly known in text input systems with custom virtual keyboards. While Quikwriting [143] and Cirrin [116] require the user to move the cursor back to the center area of a virtual keyboard after indicating the character(s) with the cursor, Huckauf and Urbina [73] presented a writing system in which a character is selected by moving with the eye to the text input field after looking at the character. Simlar as the dwell based method, this approach avoids that the user has to perform a second task apart of moving the cursor on screen. In addition,

it reduces the Midas touch problem as a separate cursor movement is need for the selection.

The last option for item selection is requiring the user to perform a secondary action apart from the cursor movement. This should further reduce the Midas touch problem. Various researchers suggest to add a second input modality, e.g. Shoemaker et al. [161] propose a system in which the Nintendo Wii Remote motion controls the cursor movement, but pressing a button results in the selection. This is convenient as the Wii Remote already contains the necessary button, but it would be quite awkward in a freehand system in which no hand-held device is present. Markussen et al. [117] use hand motions for the cursor movement, but track markers on a glove worn by the users to detect taps with the index finger. While this seems to be a natural way of selection in freehand interaction, using gloves in spontaneous interactions with public displays would be undesirable and a robust implementation using Kinect tracking without gloves is not feasible at the moment. Another solution is given by Ren et al. [148] who use a separate hand gesture for the selection. This secondary action consists of reaching with the hand for the onscreen item while still pointing at it.

Dwell based selection is already established in commercial applications, however, selection areas can provide a faster solution for the Midas touch problem with some input modalities. While avoiding the use of additional devices or special gear to achieve even faster selection with a secondary action, the pushing gestures as investigated by Ren et al. [148] promise similar advantages for freehand interaction.

2.5 User-Defined Gesture Sets

The basic rule when designing an interface is to initially define the needs of its users and gestural interfaces are no exception [134, 158, 55, 133, 154]. Nevertheless, gesture sets are often designed without sufficiently taking into accounts the preferences, habits, and needs of the actual users, and traditional user-centered approaches often only integrate the user for defining what functionality should be implemented, but how it is implemented is

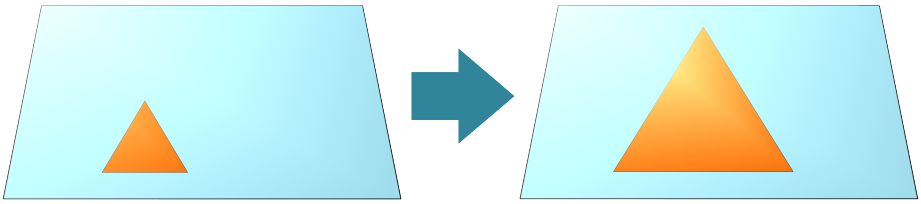
based on trial and error. This can be appropriate in cases with a very limited set of input actions or hard restrictions of the interaction technology [90]. In these cases, it is more important that the input actions can be realized robustly with the used technology and that the gesture set itself is defined in a strictly consistent and unambiguous way. However, especially with larger and more heterogeneous gesture sets, this often causes that the designed gestures are not the most intuitive ones, are inconvenient to perform or do not represent the users' natural behavior in general. This might still have been convenient for older gestural interaction technology which stayed inside the research labs. Since the Kinect brought full body interaction to the mass market, it became the time to investigate ways to provide better usability with that technology.

Several researchers started to involve the user into the design process of gestural interaction similar to the early work by Good et al. [58]. Wobbrock et al. [189] presented an approach to develop user-defined gestures for surface interfaces, and soon researchers adopted this process for other areas. For example, Kurdyukova et al. [105] used it to design gestures for transferring data between tablet computers and a multi-display environment. Ruiz at el. [152] presented results of a user-defined motion gesture set for smartphone interactions. Kray at el. [102] identified user-defined gestures that can be used to communicate a mobile phone with public display, tabletops, and other devices. Ruiz and Vogel [153] further applied the process to body gestures and tried to achieve gestures with low arm fatigue by optionally employing wrist weights. We adapt the process by Wobbrock et al. for two other application scenarios that are described in Chapter 4.

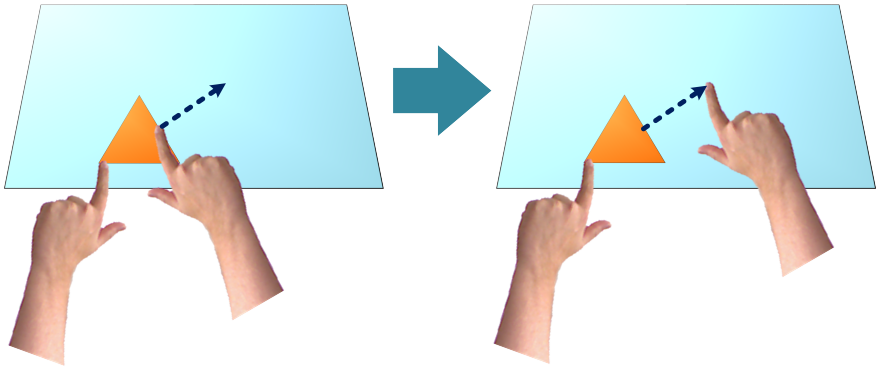
2.5.1 Method

For creating a user-defined gesture set for a given set of system actions, Wobbrock et al. [189] conduct a user study. In the study, the wanted effect within the system (i.e. system action) is presented to a participant, e.g. a small triangle is shown on the display, and then in gets bigger until a certain scale as illustrated in Figure 2.17a.

Now, the participant is asked to perform a gesture that should trig-



(a) Sample system effect



(b) Sample gesture performed by a user for the system effect

Figure 2.17: Sample system effect and user-defined gesture as described by Wobbrock et al. [189]

ger this effect. The gesture performed by the participant is recorded on video (cf. Figure 2.17b). After the study, the videos of all participants are annotated to extract the gestures performed for each system action. The annotations are used to determine gesture candidates and an agreement score for each action, which is described in detail below. In addition, Wobbrock et al. also categorized their gestures according to an own taxonomy which I already mentioned in Section 2.2. As the taxonomy does not fit well for full body interaction, I do not further investigate it here.

Gesture Candidates

The process for finding the gesture candidates is quite straight-forward:

- For each system action a, a set $M(a)$ is identified, which contains all proposed gestures.
- The proposed gestures in $M(a)$ are then grouped into subsets of identical gestures $M_i(a)$, with $i \in 1..n_a$ and n_a being the total number of

identified subsets for action a.

- The representative gesture candidate $c(a)$ for action a is identified by selecting the subset $M_i(a)$ with the largest size, i.e.:
  $c(a) = MAX_{i \in 1..n_a}(M_i(a))$

As there can be multiple sets $M_i(a)$ with equal size, there can also be multiple gestures candidates $c(a)$. The decision whether two gestures are identical or not can vary depending on the criteria that are important in the specific case. For example, in most cases, it is not important how many times a repetitive gesture has actually been repeated or whether a pointing gesture has been performed with the left or right hand. However, it is usually important in which direction the pointing actually happened.

Agreement Scores

To evaluate the degree of agreement among participants regarding the proposed gestures, Wobbrock et al. [188, 189] presented the following equation to calculate an agreement score $AS(a)$ corresponding to an action a:

$$AS(a) = \sum_{i \in 1..n_a} \left(\frac{|M_i(a)|}{|M(a)|} \right)^2$$

An agreement score $AS(a)$ is therefore represented by a number in the range $[1/|M(a)|, 1]$ with a higher value corresponding to a higher agreement, 1 representing a perfect agreement (all participants chose the same gesture for this action) and $1/|M(a)|$ representing no agreement (all participants chose different gestures for this action). The agreement score can be used as an additional measure for the quality of the gesture candidates. When there is a high agreement, the study participants had a very similar concept on how to represent the action with a gesture. Whereas with a low agreement, there was no common concept, but the participants really had to be creative to come up with an appropriate gesture for this action at all.

2.6 Supporting Users of Gestural Interaction

Even if a system has well-designed and robustly implemented gestural interaction, it can still be difficult to control for the users, as full body interaction is quite different from traditional input modalities. It is especially important to help the user in the interaction because the technology itself does not automatically provide strong (physical) affordances [65] or inherent feedback [186] as it is done by keyboard or button based devices.

A classical instrument for teaching the users on how to interact is a user manual in form of a written text book or a multimedia presentation, and those can be helpful here as well. However, it is often more efficient to have an interactive introduction, e.g. an interactive tutorial or a test scene at the first start of an application, as the user can already try out the interaction in this case [83].

It is also important to help the user during the actual interaction, as one can not assume that users are already used to this kind of interaction, and there are multiple difference in comparison to traditional input devices. At first, feedback channels known from other interaction modalities are missing, with the most important one being the haptic feedback as with physical controllers or touch screens. Therefore, users can be unsure whether their interaction was successful. Further, the beginning and end of the interaction is not implicitly given as in other modalities, in which e.g. touching the surface or taking the controller or mouse in the hand clearly starts or ends the interaction. Consequently, users are often unsure whether their interaction has already started or whether it is still running, and there is usually no straight-forward way how they can stop the interaction while still standing in front of the sensor. Last but not least, the interaction device itself offers no clear affordances on how to interact. A user can actually do any body movement in front of the sensor, but whether and how it is interpreted as an input is completely dependent on the software.

For these reasons, it is especially important for an application with full body interaction to provide mechanisms for helping the user during the interaction [195]. Those mechanisms can be categorized into **(perceived) affordances**, **feedback** and **feedforward** as done by Vermeulen et al.

[181]. (Perceived) affordances, feedback and feedforward can all be regarded as cognitive affordances in the terms of Hartson [65]. Because of a sensory affordance (e.g. visual information), the users receive information on the physical (e.g. the shape of a button invites to pressing it) or functional (e.g. a label reveals the action a button click will cause) affordances of a system. While feedback happens during or after a user action and reports results of the users' actions that might still be ongoing, (perceived) affordances and feedforward happen before the action and tell users how to perform the action (perceived affordances) or what the results of their action will be (feedforward) [181].

Using the three mentioned helping mechanisms, a full body interaction application should tell the users whether they can interact, e.g. by displaying the current state of the user tracking. The application should show what interactions are possible, so that the users know how to interact. The users should further be provided with continuous information about the progress of the input recognition and its result. The latter can include, whether an interaction has been recognized, or if not, information on what went wrong and how to solve it. Continuous information during interaction are, e.g. how close the current part of the gesturing path is to certain gesture classes, what gesture classes could still match it, or how the gesturing should be continued to complete a certain gesture.

The remaining channels for affordances and feedback/-forward in full body interaction without any additional devices except for loudspeakers or headphones and a screen are video and audio. It can be sufficient to use the video channel, especially because body actions are mainly perceived visually in human-human interaction as well. However, to amplify the effect, it can be very helpful to additionally make use of the audio channel [195]. This became quite obvious during the implementation of the freehand text input system presented in Section 5.3. Feedback through the audio channel is usually achieved by simply playing a certain audio clip whenever a meaningful input has been recognized by the system. Less often the audio is adapted, e.g. to illustrate the progress of the interaction or even to provide spatial information [145].

In the video channel, there are many more options to implement affordances and feedback/-forward mechanisms. One option present in almost every commercial full body interaction game and scientific project [15, 184] is directly displaying the user tracking image or mapping the tracking onto a virtual avatar. In the former case, the user directly sees his or her tracked shape and how the tracking skeleton is fit to it as in Figure 2.7. In the latter case, the user sees how his or her actions are recognized by the system as they are mimicked by the avatar. In both cases, the user gets immediate feedback about the current tracking state.

A common affordances and feedforward mechanism for gestural input is to display onscreen symbols with animations of possible input gestures optionally labled with the corresponding sytem effect [184]. These symbols can as well be used for feedback by highlighting them on successful recognitions. A bit more advanced feedback mechanism is to report recognition confidences [83]. When using traditional gesture recognition techniques, the recognition algorithm usually reports a score that determines how close a gesture performance was to the trained data. The score can for example be reported by displaying the bare number, a progress bar, using a color scheme or a combination of those. This helps the user to know if his or her gesture performance is getting closer to the desired, however, it does not provide information on why the performance was not correct or how it should be adapted.

For providing such information, a system needs to adapt its affordances and feedback/-forward display according to the recognition progress. In surface computing, approaches have been made to display how a user can continue towards a valid gesture, after the start of a gesture has been detected [10, 50, 103], which is visualized in Figure 2.18. The black line represents the current input path, the two colored lines visualize how the user can continue to perform a valid gesture a or b. The problem when porting this technique to full body interaction is that it would overload the video channel and distract the user, as the system does not know when the user starts a gesture and it would respond to any body movement. Delaying the information until the start of a gesture is recognized with a higher

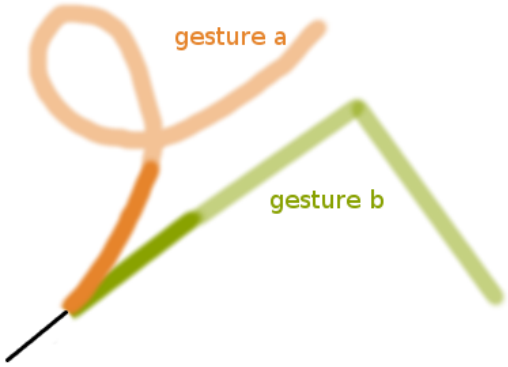


Figure 2.18: Adaptive affordances and feedback/-forward for 2D gestures as in Octopocus [10]

confidence could solve the problem, but probably postpones it to a point at which the user cannot adapt to the suggestions anymore. Therefore, new possibilities for adaptive affordances and feedback/-forward need to be found for full body interaction.

A special focus on teaching full body gestures thoroughly while correcting users when necessary can be found in the area of physical training and rehabilitation. The difference of applications for physical exercises in comparison to general full body interaction is that there is always only a single option available and it is especially important that the user performs the action very accurately. Therefore, it is at first easier to display affordances and feedback/-forward information for the current action directly on the user's avatar, as there are no other interaction options available. Furthermore, it is also very important to continuously provide that helping information for avoiding the user to get into a false posture.

For example, Anderson et al. [4] let trainers record their own full body movements. After recording the exercise, the trainers marked important steps of the performance as keyframes and they could also annotate individual keyframes with an audiovisual commentary. For learning an exercise, users went through different stages. At first, the whole exercise was played back using the recorded color image of the trainer, while keyframes were presented by speech output of a corresponding number. After that, the exercise was played pose by pose, the user's tracking skeleton was displayed and wrong positioned joints were marked with red circles. To detect the

wrong joints, the recorded keyframes were scaled to the trainee's dimensions and the euclidean distances between the joints of the trainee and the scaled skeleton were measured. In the next stage, the playback did not pause on keyframes, but ribbons leading out of the joints visualized upcoming movements. In the last stages, first the video of the trainer, and then the trainee's tracking skeleton were removed. After all except the first stage, users were provided with feedback including their overall score composed of the joint distances during all keyframes, while allowing each keyframe to be aligned by ±0.25 seconds or ±0.5 seconds on the time axis, depending on how important the timing was rated by the author of the exercise. In addition, the users could compare their recorded performance with the recording of the trainer frame-by-frame.

Velloso et al. [180] focused on arm movements. They displayed the action to be performed as a color image of a real person and the tracked skeleton of the user that should perform that action. Furthermore, indicators that looked like traffic lights visualized whether arm movements were going in the right or wrong direction, a label flashed up in case the movement was performed to fast or slow, and another label displayed the current number of repetitions.

Tang et al. [174] as well focused on arm movements. They did not visualize the actions to be performed directly, but they displayed the color image of the user during the action and tried to guide him or her towards the gesture. Therefore, they explored multiple types of affordances and feedforward during the performance of an exercise. They rendered a 2D or 3D arrow with red head and blue stem at the hand position and pointing in the currently desired direction. To reveal more of the desired movement path, they further display a brown line that concatenated a part of the upcoming path to the arrowhead. In another type, red 2D or 3D lines visualized a trace-ahead of the arm performing the exercise. Later [175], the system was enhanced with a so-called wedge visualization and a multi-camera view, i.e. a top camera in addition to the front camera. The wedge visualization can be seen as an advanced progress bar that is bent along the desired arm movement.

Zhao et al. [199] displayed a virtual character that exemplified the exercise, while a similarly looking avatar on the side of the virtual guide was controlled by the user, which is similar to Kinect fitness games such as "Your Shape: Fitness Evolved"[11]. Spheres further displayed to which position a joint should be moved during the current exercise, changed their color as soon as as they had been reached and were tagged with the number of repetitions so far. The drawback of all the presented approaches of the field of physical rehabilitation or training is that they only work when there is always only a single interaction option present.

In commercial full body interaction games, developers often try to avoid displaying gesture visualizations, but the user only needs to move an avatar and find out what to do with it on his or her own. Most games that do employ gesture visualizations, use schematic drawings that emphasize important body parts (cf. "Dance Central"[12], "Kinect Adventures"[13], ...). However, the visualizations are often only used in tutorial videos, but are hidden during the actual gameplay. Only few games, and mostly fitness games, use a fully equipped virtual character that shows how to perform certain postures or gestures (cf. "Nike Kinect Training"[14], "Your Shape: Fitness Evolved", ...).

One problem of visual affordances and feedback/-forward for full body interaction is depth perception. As the information is usually displayed on a flat screen, the depth information is missing. To solve this, one can for example show a gesture demonstration from multiple angles [195]. Other options would be the use of stereoscopic displays or render techniques that emphasize the depth perception, e.g. high-detailed textures with shadow and light calculations to provide depth cues for the human eye, or encoding the depth information in the color channel as is will be shown in Figure 5.6d.

[11] http://xbox.com/yourshapefe (accessed 2015-10-21)
[12] http://dancecentral.com (accessed 2015-9-15)
[13] http://xbox.com/kinectadventures (accessed 2015-10-21)
[14] http://xbox.com/nike (accessed 2015-10-21)

2.7 Applications for Full Body Interaction

In this section, I will look at research in multiple application areas of full body interaction, which will be investigated closer in this dissertation.

2.7.1 Virtual Keyboard Text Input

Virtual keyboard based text input systems are examples for GUI interaction that combine cursor control and item selection. These text input systems have already been largely investigated with different motion based interfaces, which are a super set of freehand interfaces. I first describe several systems using a hand-held device such as the Nintendo Wii Remote. After that, I present several attempts to implement such text input in freehand interaction as with the Kinect.

In the text entry system presented by Shoemaker et al. [161], characters were written by moving the Nintendo Wii Remote in 2D or 3D space and then pressing a button. The motion of the Nintendo Wii Remote was tracked via its infrared camera and LEDs. A comparison of a QWERTY layout, a circular layout and a 3D cube layout revealed that the QWERTY layout outperformed the other layouts both in terms of speed and error rate. Depending on the distance between the participants and the screen, the authors measured a performance of 14.5 WPM–18.9 WPM, and error rates of 2.4%–8.5% whereby the performance decreased with the distance from the screen. A similar approach was WiiNote developed by Mugellini et al. [126] that, in addition to keyboard based text input, also offered a gesture alphabet based mode. However, no data regarding the performance of this system were given in their paper. The main difference between these two approaches and our work is that the users actually have a device in their hands and that the selection of keys happens via pressing a button on this device. As I do not want to additionally employ a hand-held device, I have to explore other possibilities for the key selection.

A potential solution was given by Jones et al. [81] who solely used the accelerometer of a Nintendo Wii Remote for key selection. They avoided the need to use a button by including a selection area in their keyboard to which

users had to move back to between the characters. This way of continuous writing was in line with earlier approaches on PDAs like Quikwriting [143] and Cirrin [116] that both had the characters arranged circularly around a centered selection area. Jones et al. compared two different keyboard layouts with their approach, a "matrix layout" that arranged groups of characters in squares around a center a square, and a "tri-center layout" that arranged the characters in separate squares around three centers. In a first study, the matrix layout that was more close to Quikwriting gained a better performance with 3.7 WPM and 9% error rate in comparison to 3.3 WPM and 19% error rate for the tri-center layout. At the end of a longitudinal study with four sessions, the participants managed to improve their average speed in both layouts to 5.4 WPM. However, the users still had to hold a device in their hands.

In the approach by Markussen et al. [117], this disadvantage was slightly reduced, as users only had to wear a glove prepared with markers for tracking. The markers were tracked with an OptiTrack motion capture system. The hand motion was used to control a cursor and tapping with the index finger was used for selection. They further investigated three different keyboards: a multi tap system as used in mobile phones with nine button, a QWERTY layout, and H4 which was an own layout that had been adopted from text entry with a game controller. They measured the highest writing speeds for the QWERTY layout with 11.63 WPM, and the lowest writing speed with 4.19 WPM for H4.

Figure 2.19: Microsoft Xbox Kinect text input

The need for a hand-hold device was completely avoided in the Microsoft Xbox Kinect interface, to which Microsoft added text input with the Bing search in 2012 (Xbox Firmware 2.0.14699.0). It used a layout which arranged characters alphabetically in a single horizontal line as shown in Figure 2.19. The user had to move the hand-controlled cursor to the target

character, and while approaching it, the line of characters was zoomed to display only 10 characters for easier selection. By performing a pushing gesture in upward direction, the intended character was selected. To speed up text writing, Microsoft made use of auto-completion. The advantage of the keyboard layout was that it only requires a small area on the screen. Disadvantages were the high arm position during interaction, the permanent interruption of the cursor movement by the pushing gesture and the frequent change of the interface caused by the zoom-effect. Indeed, Hoste et al. [71] measured a rather low writing speed for the Microsoft text entry system of just 1.83 words per minute (WPM, with one word = five characters, and no auto-completion active). Together with the high error rate of partially more than five errors per sentence, there was a lot of room for improvement.

Ren et al. [148] compared three different character selection techniques and two keyboard layouts for virtual keyboard text input using a depth sensor. They compared a dwell based selection method (*Timeout*) with two methods employing hand movements in specific directions for character selection (*Reach* and *Expand&Reach*). Furthermore, they compared a standard QWERTY layout with a dual-circle layout. For the dwell-based method, they chose a dwell time of 1.2 seconds. They measured the fastest writing speed for the *Reach* method on a dual-circle keyboard layout with 8.57 WPM on the fifth day of a longitudinal study. The least error-prone method was the dwell-based one on the dual-circle keyboard with 0.00% error rate on the fifth day.

2.7.2 Interaction in Virtual Environments

In arbitrary virtual environments exist several types of in-game actions users can trigger via interaction. Those include *navigation* that serves to reach interaction possibilities, often followed by *selection* that determines the currently relevant entities before *dialogue* or *manipulation* actions are used to change the world state.

Basic navigation includes changing the position and orientation. In the simplest case, navigation control is enabled by mouse, joystick or key-

board input. But to achieve a greater amount of immersion, researchers use walking and leaning gestures [82, 110] or let the user literally walk around [41, 27]. The control schemes considered for VisTA-walk [82] include a joystick-like mapping which lets the user indicate their desired movement direction by physically stepping away from a neutral position. La Viola et al. [110] used leaning gestures for indicating the direction when traveling short or medium distances, but let the user choose their target directly by walking on a map projected onto the floor for longer distances. The use of speech for navigation is much rarer in literature, but those who apply it also tend to specify the target itself (e.g. "go to location xy") as done by Cohen et al. [29].

Corradini and Cohen [32] investigated speech and gesture inputs of users during the "Myst III – Exile" game with a Wizard-of-Oz setup. They discovered that users tend to combine both modalities and that gestures for manipulating objects mostly followed the objects' physical affordances.

For selecting objects, Van der Sluis and Kramer [178] examined how pointing gestures and verbal descriptions were combined to single out a particular option. Depending on various difficulty factors, participants focused on one channel, while the less suitable one contributed a more general, imprecise expression.

Dialogue actions are usually involved when embodied conversational agents exist in the scenario that users can speak with. In commercial games, dialogue with virtual characters is typically enabled by allowing the user to select dialogue utterances from a menu with the mouse. To allow for more human-like conversations with virtual characters, efforts have been made to support natural language input either by having the user type text using a keyboard or by making use of a speech recognizer [120, 41, 26]. Cavazza et al. [25] additionally added conversational gestures to reduce ambiguity, but still consider speech "the only practical mode of communication" since spoken words are crucial to the narrative and at least natural conversational gestures are usually ambiguous without that context.

Interactive Intercultural Training Systems

A special case of interactive virtual environments that suit well for full body interaction are intercultural training systems. Current intercultural training systems with virtual environments usually integrate virtual characters. However, such applications often only involve the user as a passive observer. As a consequence, many researchers concentrate on perception studies in which users are requested to watch scenarios with virtual characters reflecting a particular cultural background [78], [46].

Interactive applications for intercultural training often rely on dialogue enhanced by speech-accompanying gestures. Usually, natural language utterances and gestures are selected from a graphical interface. For example, Raybourn et al. [147] used drop-down mouse menus whereas Wu et al. [193] employed an interface on a PDA. While selecting gestures and phrases from a graphical interface might help people learn which gestures and phrases to use, it does not allow them to practice them. In particular, people cannot train how to perform the gestures and how to pronounce the phrases properly. Therefore, some systems additionally include speech recognition. For example, Johnson et al. [80] used a graphical interface controlled with the mouse for selecting gestures, but speech recognition for inputting utterances in their Tactical Language Training System (TLTS). So far, four versions of TLTS have been implemented: Iraqi, Dari, Pashto, and French. Users have to learn the foreign language, but also communicate appropriately with members of the simulated culture.

An application that differed significantly from the above was the ORIENT application presented by Aylett et al. [6]. ORIENT presented a large variety of interaction modalities for cultural learning based on real, physical and tangible objects surrounding the user. The interfaces included mobile phones, objects with RFID technology, a dance pad to allow for navigational actions, keyword-based speech input and symbolic gestures captured by a Nintendo Wii Remote. The ORIENT system constituted a major attempt to integrate users as active participants into a culturally sensitive application. Nevertheless, interaction was still rather cumbersome because users had to hold a device in their hands for performing gestures and the

navigation using a dance pad was little intuitive because users had to hit specific areas with their feet. As a consequence, the devices enabled active user participation, but required some training and thus might distract the user from the actual learning goal.

2.7.3 Interaction in Augmented and Virtual Reality

Augmented and virtual reality (AR and VR) are technologies for visually immersing the user into a virtual world (VR) or enhancing the real world with virtual objects (AR). To provide a fully immersive system, interaction in AR and VR environments should be immersive as well, so they should be as close as possible to natural interaction in real life [14, 192, 120, 77]. Depending on the interaction task, immersive interaction modalities are speech input for dialogues, body gestures and movements for conversational gestures and navigation, and physical interaction for manipulating objects. I define physical interaction as interaction in which a (real or virtual) object is (virtually) touched with the body or handheld devices to manipulate its position, orientation or form [77, 144, 66].

Current technology usually has various restrictions for applying physical interaction, e.g. objects cannot be rendered on a head-mounted display in a distance close enough for touch interaction, the system cannot render the occlusion of hands on virtual objects because of missing depth information, and it is hard to provide haptic feedback when touching virtual objects. An alternative that still provides immersive interaction close to real life interaction is to use body gestures for manipulating objects. In this case, the object is manipulated remotely while performing the same body motion as if touching it.

2.7.4 Controlling Machines or Robots

Moving away from the virtual world, full body interaction can as well be a good choice for controlling arbitrary devices in real-life. While freehand GUI interaction suits well to replace remote controls for radios, TVs or

other media devices, full body gestures can be a good option to naturally command robots or machines.

Nguyen-Duc-Thanh at el. [131] demonstrated an approach to control a humanoid robot (Nao[15]) using human body language. Their method was based on a Semaphore alphabetical system in which the human body poses and gestures, performed with a Kinect, were recognized as alphabetical characters that could be interpreted by the robot. Broccia et al. [19] used Kinect to recognize the users upper body movements, which were mimicked by a humanoid robot based on a mathematical mapping of the human movements to the robots joints. For navigational purposes they used full body gestures like stepping forward and backward or turning the body. Cabibihan et al. [22] conducted a study to investigate the recognition of 15 human-like gestures performed by a human actor and an anthropomorphic robot. Their results revealed that eight gestures were recognized from the human-actor's video and six gestures were recognized from the robot's video. Their work addressed what robot gestures humans could understand, which is the opposite to defining gestures to control a robot. Some work followed multimodal approaches, mostly combining speech with gesture commands [166]. Other work efforts were put towards controlling robots using pointing gestures [156], but such methods were limited to a certain range of commands. Moreover, Hu et al. [72] developed simple hand gestures for robot navigational actions, while Konda et al. [99] employed full body postures.

Quite similar to controlling humanoid robots is the control of industrial machines, which can as well be done with full body interaction. For example, Shirwalkar and Singh [160] or Lambrecht et al. [107] used hand motions to naturally control a industrial robotic arm, while Stipancic et al. [167] used multiple body parts.

[15] http://aldebaran-robotics.com (accessed 2015-10-23)

Chapter 3

Exploration of Full Body interaction

In this chapter, I will explore full body interaction as an input modality. I will investigate, how users interact with systems via full body interaction. Accordingly, the next chapter compares different ways on how to use full body interaction for completing interaction tasks in an interactive storytelling scenario. Afterwards, I will look at additional interaction tasks and explore whether users prefer speech or gestural input to trigger those tasks.

3.1 Body Gestures versus Freehand GUIs

Our first application applying full body interaction was presented in [98]. The application represented an interactive storytelling scenario that was enhanced with full body interaction for two players. We implemented two types of full body interaction and compared them in a user study: **full body gestures** that directly triggered the actions (cf. Section 5.4, here called (*gesture mode*)) and **freehand GUI interaction** that used a cursor to select UI items which represented the actions (cf. 5.3, here called (*button mode*)). In the *button mode* the position of a body part (the hand) was continuously applied to a virtual object (the cursor). The *button mode* as

well included discrete events for selecting GUI items, however, the corresponding "body gesture" was – as commonly implemented – a very small hand movement or even a simple resting of the hand, but no actual set of gestures. In contrast, our *gesture mode* waited that a certain body gesture out of a gesture set was performed by the user and then discretely triggered an event in the system.

Various approaches for providing innovative interaction modalities in interactive storytelling have been investigated in the past, although the application of full body interaction is rare (cf. Section 2.7.2). The first Kinect games available on the Microsoft Xbox 360 console applied full body interaction, however, they mainly comprised sport and fitness games (e.g. *Kinect Sports*), racing games (e.g. *Kinect Joy Ride*), and party and puzzle games (e.g. *Game Party in Motion*). Most of the interaction in those games was similar to our *gesture mode*, while an interaction similar to our *button mode* was only used in graphical menus when the actual game stayed paused. In opposite to those games, we wanted to use the novel full body interaction within an application that concentrates on a story, i.e. an interactive storytelling scenario.

3.1.1 Game Books and Interactive Storytelling

To facilitate the authoring of the story, we based it on the **game book** "Sugarcane Island" by Packard [137]. Game books can offer a well-written and non-linear story well-suited to an interactive storytelling scenario. An early example of the game book genre was the short story *An Examination of the Work of Herbert Quain* that included a novel made up of thirteen chapters and covering nine different story lines. This was achieved through the fact that the first chapter could lead to one of three subsequent chapters, and each of those in turn had three possible subsequent chapters. "Sugarcane Island" by Packard [137] was a more current instance of the game book genre, but included the same mechanisms. It was the first book in the popular *Choose Your Own Adventure* series and started with a shipwreck on an expedition. Waking up on the beach of an unknown island, the reader needed to find a way to survive. After each text section of the book, the

reader had to decide how to proceed. The given choices referred to different pages in the book to read next. For example, the reader had to make a decision on page 17 of the book[1] in the following way:

Page 17: You wake up in a thatched hut. [...] You take a peek outside and observe ferocious looking natives doing a tribal dance around a fire.
You decide to flee. Go to *Page 27*.
You stay. Go to *Page 28*.
Page 27: You start up and sprint into the woods. [...]
Page 28: A little while later, some natives appear in your hut. [...]

Therefore, Game books can offer clear points in the story, in which the user has to interact by deciding on how to go on, which makes game books a good basis for and interactive storytelling scenario.

3.1.2 Implementation

Our application ran on the *Horde3D GameEngine*[2]. Additionally, we used *SceneMaker 3* [124] to model and execute the story as a hierarchical finite state machine extended with multimodal scene scripts. The scene scripts consisted of the text to be spoken and embedded additional commands such as the playback of animations or sounds.

Unlike the book, our application was designed for two users listening to a virtual narrator and interacting at specific points to influence the story. As in the book, the users had to decide on how to go on at specific points in the story as described in the preceding section. We integrated those decisions in a Wizard-of-Oz design with speech commands and evaluated it in a user study. As this does not concern the topic of this dissertation, I will omit this part here, however the reader can look it up in the original publication [98]. Nevertheless, a comparison of speech and gesture input with real-time recognition will follow in Section 3.2.

As a second type of interaction, we added so called **quick time events (QTEs)** that are frequently used in current video games. To our knowledge, there had been no scientific studies about the application of QTEs,

[1] Re-translated to English from the German version [137]
[2] http://www.hcm-lab.de/projects/GameEngine (accessed 2015-9-15)

until the publication of this work [98] . However, good examples of games, which make extensive use of QTEs were *Fahrenheit (Indigo Prophecy)* and *Heavy Rain* from the games developer Quantic Dream[3]. Whenever a QTE occured in a video game, a symbol representing a specific action on the control device appeared on screen. The user then had a limited amount of time to perform that action in order to complete the QTE successfully. Most times, successful performance of the QTE resulted in a particular action by the player avatar, while unsuccessful performance caused the player avatar to fail in this action. In general, the utilization of QTEs can range from enriching cutscenes with interactivity to using QTEs as the main gameplay mechanic. In opposite to the video game examples, we did not use traditional control devices, but full body interactions.

Some passages of the book already contained situations that were well-suited for the application of QTEs. One example in the original text (p. 13)[4] read as follows:

You start to climb the steep hill. It is highly exhausting and one time you loose your grip and almost fall down a rock face. But finally you arrive at the top.

The modified text part was (modifications marked with bold font):

You start to climb the steep hill. It is highly exhausting and one time you ***almost*** *loose your grip.*

The QTE started immediately after this text, and when it was solved, the following message was narrated:

You manage to hold on just in time and *finally you arrive at the top..*

Otherwise, the text was:

You fall down a rock face but you are lucky that you did not get hurt too badly.

If the QTE was solved, the story continued as in the original version (*here:* page 20), but if not, it jumped to a different, but appropriate, page of the story (*here:* page 14).

As soon as a QTE started in our application, a countdown appeared

[3] http://www.quanticdream.com (accessed 2015-9-15)

[4] Re-translated to English from the German version [137]

centrally in the upper part of the screen with a symbol shown for each user representing the action requested (see Figure 3.1). At the moment one user solved a QTE, a mark was shown on top of the symbol, providing immediate feedback. The full QTE was solved if both users successfully completed their action before the countdown reached zero.

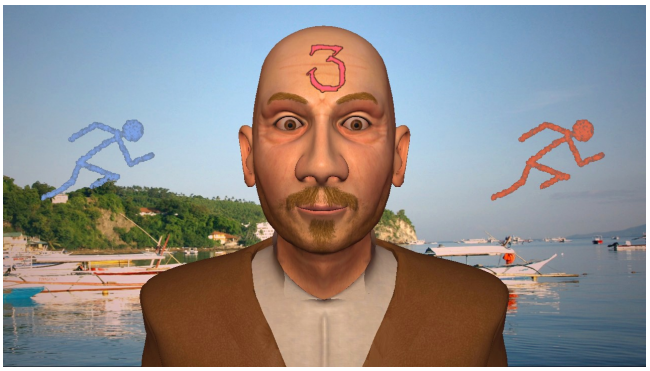


Figure 3.1: Screenshot of the quick time even (QTE) "Run" in Sugarcane Island

We employed two different modes to carry out of QTEs, both using full body interaction: In the first mode (i.e. the *button mode*), each user had to press a randomly-positioned and -sized button on screen, using a cursor controlled by moving the hand. In the other mode (i.e. the *gesture mode*), users needed to perform gestures that were indicated on screen via one of the symbols shown in Figure 3.2.

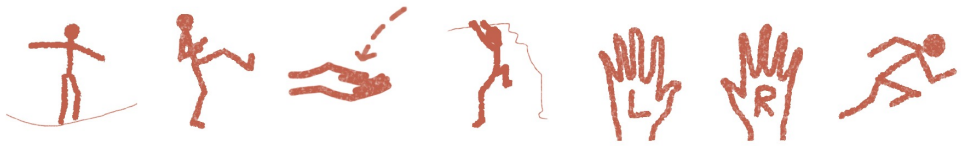


Figure 3.2: QTE symbols in Sugarcane Island

The *button mode* was implemented as freehand GUI interaction as described in Section 5.3. The user had two ways to select a button: by hovering a button for more than 1.5 seconds with the cursor, or by performing a push gesture in direction of the screen. As soon as the QTE started, a button containing text for the requested action was shown for

each user. The buttons were positioned randomly: on the right half of the screen for the right user, and on the left half for the left user. In addition, they randomly had a slightly different size for each QTE. Once a button was activated, it disappeared and a tick appeared to inform the user about the successful action. Overall, the button mode represented the action of the QTE in an abstract way. The meaning was provided by the text string, but the actual interaction task depended on random parameters only.

In the *gesture mode*, the requested gestures represented the QTE actions more directly. Figure 3.2 displayed the symbols used to visualize the requested user actions (from left to right):
Balance: Hold hands out at shoulder-height; *Kick*: Perform a kick with one leg; *Catch*: Put the hands together in front of the body; *Climb*: Move hands up and down in front of the head, as if climbing; *Left and right Hand*: Raise the left or right hand; *Run*: Move the feet up and down like running, but without moving.

Figure 3.3: QTE gestures performed by two users in Sugarcane Island

Figure 3.3 shows two users in front of a screen, performing the QTE gestures “Kick” (left user) and “Catch” (right user). Note the Kinect placed

centrally below the screen. The left user has already succeeded in the QTE "Kick", so a green tick has appeared over the corresponding symbol.

The recognition for the gestures shown in Figure 3.2 was implemented according to Section 5.4 using an early version of the FUBI framework. For example, "left hand" was implemented using the position of the left hand relative to the left shoulder and testing whether the left hand was currently above the shoulder (y-coordinate of the hand greater than y-coordinate of the shoulder). Other more complex gestures were modeled as simple finite state machines concatenating postures to sequences with specific time constraints in an early version of the combinations described in Section 5.4.

3.1.3 User Study

In a user study, our intention was to investigate whether the *button* or *gesture mode* would be preferred, and to initially test how well the recognition of full body gestures can be implemented with simple rule-based recognizers and additional finite state machines for concatenating them to sequences.

18 participants were involved in the study with an average age of 24.7 years. The participants were arranged into groups of two. The two participants in a group had to stand in front of a 50 inch plasma display at a distance of between 1.5 and 3 meters. The Microsoft Kinect was installed slightly below the display.

We applied the "within subjects" design, and therefore each group had to participate in one application run of both conditions (i.e. button and gesture mode). To prevent positioning effects, we counterbalanced the order in which the conditions were encountered. An application run consists of two parts, a short introduction and the main story part. The short introduction was intended only to familiarize the participants with the interaction with the system. After each run, the participants had to fill out a questionnaire that was derived from the IRIS Evaluation Measurement Toolkit[5]. Each statement was given on a nine-point Likert scale ranging from "strongly disagree" (-4) through "neutral" (0) to "strongly agree" (4).

[5] http://iris.scm.tees.ac.uk (accessed 2011-11-30)

3.1.4 Results

We applied two-tailed paired t-tests to validate our results. The questionnaire responses showed that the participants found the gesture-based QTEs (M = 2.9, SD = 0.9) significantly easier to use than the button-based events (M = 2.0, SD = 1.9), where they had to simply point at a specific button label ($t(17)$ = 2.1, $p < 0.05$, r = 0.29). The participants also would imagine that most people are able to learn the system with the gesture-based QTEs (M = 3.3, SD = 0.8) significantly quicker than the one without (M = 2.4, SD = 1.5; $t(17)$ = 2.2, $p < 0.05$, r = 0.35). The *gesture mode* (M = −3.1, SD = 1.0) was considered more comfortable to use compared to the *button mode* (M = −1.3, SD = 2.1; $t(17)$ = 3.5, $p < 0.01$, r = 0.48). The participants were significantly more satisfied with the *gesture mode* (M = 2.3, SD = 1.0) than with the *button mode* (M = 1.6, SD = 1.4; $t(17)$ = 2.4, $p < 0.05$, r = 0.28). The *gesture mode* (M = 0.0, SD = 2.5) was also considered as significantly less inconvenient compared to the *button mode* (M = −2.3, SD = 1.7; $t(17)$ = 4.0, $p < 0.001$, r = 0.47). Interaction in *gesture mode* (M = 3.1, SD = 1.0) was experienced as significantly more fun than in *button mode* (M = 1.0, SD = 2.3; $t(17)$ = 3.8, $p < 0.01$, r = 0.51), and lastly the participants stared at the screen with significantly higher expectations in *gesture mode* (M = 1.4, SD = 1.6) compared to the *button mode* (M = 0.8, SD = 1.5; $t(17)$ = 2.2, $p < 0.05$, r = 0.19).

The recognition within the button and gesture mode worked very well. The participants succeeded in 93% of all actions within the button-based QTEs (i.e. 67 out of 72). For the gesture-based QTEs the participants were even more successful, with 97% of all possible actions (i.e. 65 out of 67).

3.1.5 Conclusion

In our study, the *gesture mode* was preferred to the *button mode* according to multiple evaluation criteria. The more natural gestures not only supported better usability, but also resulted in greater comfort and made more fun. Furthermore, we also showed participants were better at solving QTEs using natural gestures than the cursor-button interaction. This in-

dicated, that it would be worthwhile designing new and more natural ways of interaction for a full body tracking system, instead of adapting the conventional point-and-click paradigm. Therefore, we emphasized our focus on implementing gesture recognition capabilities in the FUBI framework. Nevertheless, depending on the requirements of the application and its range of possible inputs, freehand GUI interaction can still be a good option, as it will be shown in later sections. As the recognition using simple rule-based recognizers already showed a good accuracy, we used this kind of gesture recognition in the FUBI framework, but later enhanced it with many other techniques to cover a wide range of application scenarios.

3.2 Speech versus Gestural Interaction

When applying full body interaction with the goal of achieving a natural form of user input, it is standing to reason to consider another natural interaction modality as well that is speech input. Speech and body motions serve as the main interaction modalities in the real world, and therefore, it seems quite logical to use them for immersive interaction in virtual worlds as well. This is emphasized by the fact that the Kinect as well as some of the other depth sensors additionally include microphone arrays. However, the different interaction modalities need to be harmonically integrated with the virtual setting and intuitive for the user. In consequence, most consumer products at the time of writing this thesis only used speech or gesture functionality to enhance a specific type of interaction, whereas they still relied on traditional input devices for other types or automate parts of the interaction, e.g. in the racing game "Kinect Joy Ride"[6], the player used hand motions for steering to the left and right, but the car accelerated automatically, and in the role-playing game "Mass Effect 3"[7], the Kinect microphone was used for speech commands, but the rest of the input happened with a game pad instead of using gestural interaction provided by the Kinect depth sensor.

[6] http://xbox.com/kinectjoyride (accessed 2015-9-15)
[7] http://masseffect.bioware.com (accessed 2015-9-15)

In this section, I describe a system in which we solely used gesture and speech interaction, previously presented in [79]. In the corresponding study, users could always choose between these two modalities and we investigated, which modality was chosen for which interaction task.

In contrast to most of the related work regarding human-like interaction in virtual environments as described in Section 2.7.2, our application included all of the four interaction tasks: **navigation**, **selection**, **manipulation**, and **dialogue**. In addition, the system actually applied real-time recognition of inputs as opposed to a Wizard-of-Oz setup. Instead of investigating different implementations of one modality or examining multimodal usage, our goal was to determine the primary interaction modality for each of those tasks. For this purpose, we conducted a study as described in Section 3.2.2.

Figure 3.4: User interacting with our application with speech and gestures

3.2.1 Implementation

Our system displayed a virtual world in a first person perspective on a 50 inch screen using the Horde3D GameEngine as depicted in Figure 3.4 similar to the application presented in the preceeding section. Each action in our system was linked to both a gesture and a speech command which

could be used interchangeably. Gesture recognition was implemented using the "Full Body Interaction Framework" (FUBI) in combination with a Microsoft Kinect for Xbox 360 placed centered below the screen, and using the OpenNI framework and PrimeSense NiTE middleware for user tracking. Speech was processed with the Microsoft Speech Platform[8] for multi-keyword spotting on the audio of a wireless headset's microphone.

Our scenario consisted of actions belonging to the four different tasks *navigation*, *selection*, *dialogue*, and *manipulation*. Users needed to navigate to various selectable entities, and then performed dialogue and manipulation actions on them before moving on. In total, one had to perform about 17 actions per task to complete the scenario. For performing an action, users could always choose between speech or gesture input and our primary hypothesis was that the two modalities would not be equally suitable for every task. The implemented inputs are explained in the following.

Our application used a *navigation* vocabulary for basic movements (i.e. move left/right/forward/backward) and rotations (i.e. turn left/right/up/down), which was considered closer to reality and more flexible than indicating a target directly, which would also overlap with the selection task. Gesture input for movements was based on a walking metaphor similar to the joystick control scheme by Kadobayashi et al. [82], e.g. users had to step forward for starting a movement to the front. Similarly, the rotations directly used the torso orientation, e.g. users actually had to turn left for starting a rotation to the left and they had to lean backwards for tilting their viewing angle upwards, which also resembled the rotation commands described by LaViola et al. [110]. Feedback for the movement was provided by an icon (see Figure 3.5 on the upper left) that shows the user's physical position relative to a neutral zone defined as a 40 cm × 40 cm square about two meters in front of the screen. For speech, navigation commands consisted of naming their type and direction, e.g. saying "turn left" for turning left, or "forward" for moving forward. A label below the movement icon displayed the recognized navigation command for feedback. For this task, our hypothesis was that gestures would be preferred to spoken commands

[8] http://msdn.microsoft.com/library/hh361572.aspx (accessed 2015-9-15)

as they are closer to real-life navigation.

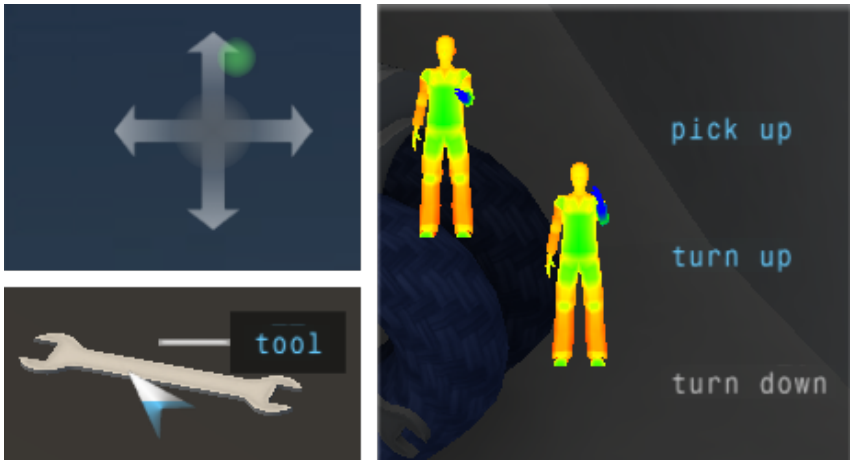


Figure 3.5: GUI for speech and gestures – *Upper left:* Movement icon; *Lower left:* Object selection; *Right:* Available (modification) actions for the object on the lower left-hand side

Interactive objects and characters in our scenario were marked with labels which were colored blue instead of white when they were reachable. Pointing gestures were used to move a cursor across the screen and the user had to hold it above an entity for 0.5 seconds for *selection* (dwell-based selection), during which the cursor filled up with color as shown in Figure 3.5 on the lower left. This was similar to the "button mode" described in Section 3.1. The same selection is performed by speaking the entity's name as shown on its label, which was kept unambiguous in our scenario. Either command results in the display of available interactions (manipulation or dialogue) for this entity, presented in the style of a context menu. As both modalities seemed equally natural for selection, we did not have a clear hypothesis for this task.

For virtual characters, the context menu displayed sentences which could currently be spoken to them. Fifteen unique phrases were available throughout the scenario, each of which contained one or more semantically important keywords (colored in blue) which needed to be said in the given order for speech input. The remaining words (colored in white) were optional and could be changed or omitted by the user. This approach resembled the one

described by Cavazza et al. [25]. For applying gestural interaction to the *dialogue* task, we were again using the pointing gestures as in the selection task. Therefore, the desired sentence was chosen by moving the cursor to a button-like target next to it. The first reason for this decision was that conversational gestures were often ambiguous if used without accompanying speech as stated by Cavazza et al. [25]. Furthermore, not every topic had a straightforward gesture representation, e.g. the scenario's very first question of "Where am I?" would be hard to express with a single gesture. As speech seemed to be a very obvious choice for dialogue, we hypothesized it to be preferred for this task.

Interactive objects could be *manipulated* by gestures which resemble real-world actions as suggested by Corradini and Cohen [32], e.g. raising the knees was used to step onto a bed, and moving the hand like pulling a lever was used for actually doing this. Animated human figures displayed the motions that are expected from the user as depicted in Figure 3.5 on the right-hand side. These animations were automatically generated from the same XML gesture definitions used by the FUBI framework for gesture recognition. Based on the given speed limits, state durations and transition times, movement paths for the joints of a virtual character were defined and later applied using inverse kinematics. The speech alternative mainly consisted of the action's verb, but occasionally, a second parameter such as a tool or direction was added for clarification, e.g. "turn up" is used for turning a spanner upwards. All currently available speech commands were listed in blue next to the animated figures for the corresponding gestures, whereas actions which might become available later were grayed out. Overall, 14 different keywords and 18 different gestures were included for the manipulation task. The hypothesis for this task was that gestures would be preferred, as they were closer to object manipulation in real life.

3.2.2 User Study

Twelve participants (eleven male, one female) were recruited at our university campus. Their age ranged from 24 to 35 years (M = 29.5), all were right-handed, and either native speakers or fluent in German. Seven had

rarely used speech input before (0–10 times) whereas five were rather experienced with it (used > 10 times or regularly). All were familiar with motion-based interaction (used > 10 times or regularly).

They were first introduced to the various controls and could practice them in a simpler virtual setting. Therein, the users were motivated to test both modalities for all four tasks. This introduction took about five to ten minutes. Afterwards, they played the main scenario which lasted about 20 minutes, and they were free to choose either modality for any interaction they encountered. After completing the scenario, the participants filled in a questionnaire which asked for their preferred modality and their opinion on both input options for each task. The latter was done by rating the following statements on a five point Likert scale ranging from 1 (completely disagree) to 5 (completely agree): "It was difficult to recognize or remember the commands for the desired action", "the commands for these actions felt natural", "it was tiring to give the commands" and "the recognition worked reliably". In addition, recognized commands were automatically logged along with the chosen modality and the task they belonged to.

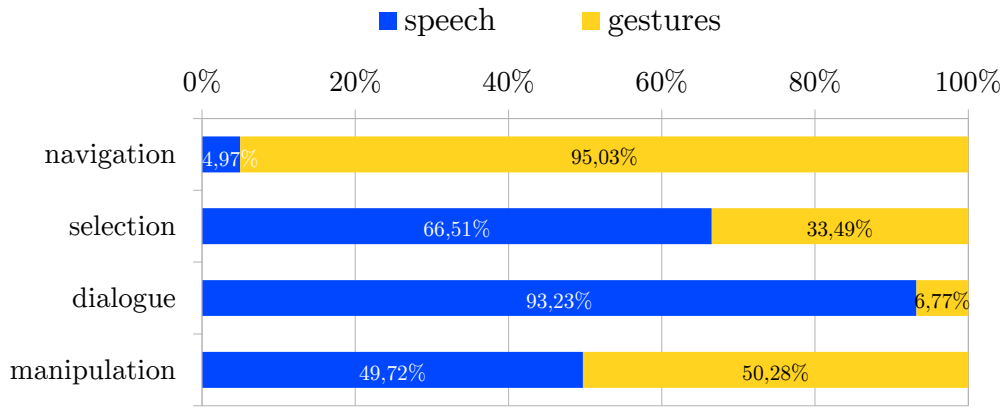


Figure 3.6: Average speech and gesture usage per interaction task

The average modality usage in the four interaction tasks is depicted in Figure 3.6. Our primary hypothesis that modalities would not be equally suitable for each task was confirmed by a Friedman's ANOVA (used as parts of the data were non-normally distributed) which showed that participants used different ratios of gesture and speech inputs for them ($\chi^2(3) = 30.18$,

$p < 0.001$). In particular, Wilcoxon signed-rank tests (with a significance level of 0.0125 for Bonferroni correction) showed that a significantly higher percentage of gestures was used for navigation than for the three other tasks ($T = 0$, $p < 0.0125$, $r = -0.62$) and a significantly higher percentage of speech was used for dialogue compared to the other tasks ($T_{manipulation} = 1$, $T_{selection} = 0$, $p < 0.0125$, $r_{manipulation} = -0.61$, $r_{selection} = -0.54$).

3.2.3 Results

For each of the tasks, Wilcoxon tests were used to compare average usage and user ratings between speech and gestures. In the dialogue task, participants used significantly more speech utterances than gestures ($T = 0$, $p < 0.01$, $r = -0.90$). Speech was further rated as significantly less difficult to learn ($M = 1.08$, $SD = 0.29$) than gestures ($M = 2.25$, $SD = 1.22$; $T = 0$, $p < 0.01$, $r = -0.74$), it was considered more natural ($M = 4.92$, $SD = 0.29$) than gestures ($M = 2.83$, $SD = 0.94$; $T = 0$, $p < 0.01$, $r = -0.86$), less tiring ($M = 1.17$, $SD = 0.39$) than gestures ($M = 2.58$, $SD = 0.90$; $T = 0$, $p < 0.01$, $r = -0.83$), and more reliable ($M = 4.83$, $SD = 0.39$) than gestures ($M = 3.67$, $SD = 0.78$; $T = 0$, $p < 0.01$, $r = -0.79$). In the navigation task, we got a significantly higher usage of gestures than speech ($T = 0$, $p < 0.001$, $r = -0.99$) and a lower difficulty rating for gestures ($M = 1.42$, $SD = 0.67$) than for speech ($M = 2.50$, $SD = 1.24$; $T = 10.5$, $p < 0.05$, $r = 0.59$). We found no significantly different modality usages in the manipulation task, but a significantly better user rating for speech that was rated as less difficult to learn ($M = 1.25$, $SD = 0.45$) than gestures ($M = 2.67$, $SD = 1.07$; $T = 0$, $p < 0.01$, $r = 0.78$), less tiring ($M = 1.33$, $SD = 0.49$) than gestures ($M = 2.25$, $SD = 1.14$; $T = 0$, $p < 0.05$, $r = 0.70$), and more reliable ($M = 4.92$, $SD = 0.29$) than gestures ($M = 3.83$, $SD = 0.94$; $T = 0$, $p < 0.01$, $r = 0.75$).

The stated modality preferences are again in favor of gestures in the navigation task (11 preferred gestures, 1 preferred speech) and of speech in the dialogue task (preferred by all 12). Furthermore, they indicate a preference for speech in the selection task (7 preferred speech, 1 gestures, 4 were undecided), but an equal distribution for manipulation (5 preferred speech, 5 preferred gestures, 2 were undecided).

3.2.4 Discussion

For navigation, our hypothesis in favor of gesture input was confirmed by its higher usage and stated preference, as well as the fact that gestures were rated as easier to learn. This was in line with Kadobayashi et al. [82] who considered walking gestures to be more intuitive for navigation than using a mouse. However, there might be different results when using a navigation approach with direct target selection.

For the selection task, we found no significant differences, only the stated preferences indicate a tendency for speech. One reason might be distinctions between the selection targets, as three participants mentioned that they liked to reach for an object with their hands whereas two preferred addressing characters by speech. Different sizes and placements of the objects might have further influenced the modality choice, as some objects were more difficult to point at than others, similarly observed by van der Sluis and Krahmer [178].

The hypothesis that speech would be preferred for dialogue as derived from Cavazza et al. [25] was clearly confirmed. All participants named it as their preferred modality, it was used most of the time with nine participants even using it for every single sentence, and the user ratings were very positive with all items close to the extremes. Apart from this clear result, it has to be mentioned that there exist dialogue utterances that can be naturally represented by gestures, e.g. nodding for "yes" or a greeting gesture for "hello", but this is not the case for arbitrary sentences.

We assumed a preference of gestures for the object manipulation task, but this hypothesis could not be confirmed as both modalities were used with almost equal preference and the user ratings were even in favor of speech. A similar variety of modalities was observed by Corradini and Cohen [32], who additionally reported that users preferred to use both, gestures and speech, in a multimodal way.

Hints for another explanation could be found in the study, as the users seemed to follow two different behavior patterns. Speech users seemed to be more focused on progressing, often calling the actions as soon as they appeared on screen, instead of first watching the gesture animations to

figure out how to perform them. On the other hand, gesture users seemed to perform the task in a consciously more natural way and some also exhibited role-playing behavior such as worrying about being heard by the virtual characters. Therefore, interaction designers should investigate their target group's preferences and decide between a more natural and engaging object manipulation using gestures or a faster one using short speech commands.

3.2.5 Conclusion

In this section, I examined which modality users preferred regarding four main interaction tasks in a virtual environment. We conducted a study on a system in which we successfully implemented all four interaction tasks with real-time recognition for both speech and body gesture input using low-cost technology. It was confirmed that a gestural walking metaphor suits navigational tasks better while speech was chosen for dialogues. For selection and manipulation, no clear preference was obtained, but we observed possible reasons for the different modality choices between the users. While speech is usually faster to perform for discrete interactions as selecting an item or a sentence. Gestural interaction is better suited for continuous interaction in which ongoing inputs are needed within a certain duration to adjust the wanted outcome, as for navigation. Nevertheless, gestural interaction also offers advantages for discrete interaction tasks, as not in all scenarios, it is possible or wanted to use speech, e.g. in noisy environments, for privacy reasons, or when the bodily activity is part of the application's targets as in fitness games.

Chapter 4

User-Defined Full Body Gestures

In the applications of the last chapter, the gestures were chosen by the developer of the system according to his or her own preferences. Nevertheless, it might be that they were not the most intuitive ones for the actual users. The goal of this chapter is to better integrate the user in the design process. Therefore, I adopt and modify the process by Wobbrock et al. [189] as described in Section 2.5. In opposite to Wobbrock et al. who investigated surface gestures, my goal is to identify intuitive gestures for applications with full body interaction. I use their definition to calculate an agreement score. However, I enhance their process (see Section 2.5.1) for finding gesture candidates by allowing multiple levels of candidates. Therefore, I do not only look at the largest subset of identical gestures $M_i(a)$, but I propose to order all of the subsets for getting alternative candidates in the case the first candidate cannot be used, e.g. for technical reasons. In this way, I define multiple gesture candidates c_j in the following way:

$$c_j(a) = MAX_{i \in 1..n_a, M_i(a) \neq c_k(a) \forall k<j}(M_i(a))$$

As not all alternative gesture candidates c_j are similarly often represented in the set $M(a)$, I propose that an alternative candidate should only be taken if its size is not much smaller than the size of the first candidate, e.g. one could define that an alternative is only taken into account if its

size is at least half the size of the first candidate.

Before applying the design process, I will develop an own taxonomy for full body interaction in Section 4.1. The taxonomy will be used for categorizing gestures in the two preceding sections that present two cases, in which we completely went through the design process for creating user-defined full body gestures. The first study creates a gesture set for controlling a humanoid robot, and the second study investigates input gestures for the intercultural training system Traveller.

4.1 Taxonomy for Full Body Interaction

As already could be seen in the preceeding sections, full body interaction offers many possibilities for interacting with a computer dependent on which body parts are used, whether it is discrete or continuous, how the inputs are interpreted, what effect they have within the system, etc. In the next section I will describe different types of full body interaction that are investigated in this thesis. Afterwards, I will develop an own taxonomy of full body gestures, as existing ones are not perfectly suiting (cf. Section 2.2).

4.1.1 Types of Full Body Interaction

One key property on which full body interaction can be categorized is whether it is discrete or continuous. Continuous full body interaction means, that while the user is interacting, e.g. moving the hand from left to right, the system continuously interprets input signals and changes the system's state, e.g. a cursor is moving from left to right on the screen dependent on the hand movement. In discrete full body interaction, the user first finishes the interaction, e.g. performing a circle gesture, and only afterwards, the input is interpreted and results in a change of the system's state, e.g. an item on the screen is selected. Continuous full body interaction can further be categorized according to what the interaction is controlling within the system and dependent on that, which body parts are used for the interaction. In the case that continuous full body interaction really uses the whole body and its tracked three-dimensional positions/ori-

entations, its input is usually used to control some kind of avatar on the screen. I will further investigate this type of interaction in Section 5.2. In the case that continuous full body interaction only uses the hands, the input is usually used to control a cursor or something else that indicates a position on the screen or in the virtual world. This type of interaction is heavily used in freehand GUI interaction that I will further investigate in Section 5.3. Nevertheless, freehand GUI interaction also needs a discrete interaction as in the example above, in which the selection of an interface item could be elicited with a circle gesture. The interpretation of continuous interaction is usually straight-forward as the used information, i.e. joint positions and orientations only need to be mapped to virtual positions on the screen while additionally applying some filtering on them. In opposite to that, discrete full body interaction requires more interpretation of the system, as it has to analyze the data stream for spotting and recognizing gestures performed by the user that are meant as input to the system. Accordingly, this kind of interaction is the main part of the Full Body Interaction framework and will be described in Section 5.4.

4.1.2 Taxonomy for Full Body Gestures

Similar to natural gesturing, body gestures that are meant as an input to the computer can further be categorized according to multiple properties. In this section, I describe our taxonomy for full body gestures which is used throughout this thesis.

I define three dimensions for the taxonomy of full body gestures: *form*, *gesture type*, and *(involved) body parts*. Each dimension consists of multiple items, shown in Table 4.1. The dimensions are partly based on the Taxonomy used by Wobbrock et al. [189] and adapted to match full body gestures. However, the gesture type dimension is closer oriented at McNeill [122]. We introduced this taxonomy in [93], however, the gesture type dimension (previously named nature dimension) of the current version is extended by emblematic gestures.

Form distinguishes between static and dynamic gestures (without and with movement respectively). Static gestures have a preparation phase at

Table 4.1: Taxonomy of full body gestures

Form	static	A static body posture is held after a preparation phase.
	dynamic	The gesture contains movement of one or more body parts during the stroke phase.
Gesture Type	deictic	The gesture is indicating a position or direction.
	iconic	The gesture visually depicts an icon and directly represents a real-world property.
	metaphoric	The gesture visually depicts an icon and describes a real-world property in an abstract way.
	emblematic	The gesture is an artificial symbol that does not represent a real-world property, but represents meaning, which needs to be learned and is often culture specific.
Body Parts	one hand	The gesture is performed with one hand.
	two hands	...with two hands.
	full body	...with at least one other body part than the hands.

the beginning, in which the user moves into the gesture space, but the core part of gesture is after the preparation phase. Therefore, the gesture is kept for a certain amount of time before the user releases it again in the retraction phase. In opposite, dynamic gestures have a clear stroke phase including the movement of specific body parts between the preparation and retraction phases.

The *gesture type* dimension is oriented at the taxonomy by McNeill [121]. It uses four of McNeill's categories in the following way: Deictic gestures indicate a position or direction. These gestures can be either static, e.g. pointing to the right, or dynamic, e.g. waving to the right. They can be performed with one hand, two hands, or even other body parts, e.g. tilting the head. Iconic gestures convey information by visually depicting an icon that directly represents a physical, spatial or temporal property of a real-world referent. Especially for full body gestures, iconic gestures are often

very direct and a real-world action is described by actually performing it like a pantomime, e.g. walking in-place for representing walking. Metaphoric gestures visually depict an icon as well. However, they describe the real-world property in a more abstract way. An example is performing a circular movement with one hand which depicts an accelerating wheel and has the meaning that one should walk faster. Emblematic gestures are artificial symbols that do not represent a real-world property, but represent meaning, which needs to be learned and is often culture specific. Examples are the thumbs-up sign or a head nod, which both can have positive or negative meanings depending on the cultural background.

The *body parts* dimension should be self-explanatory. It distinguishes between one hand, two hand, and full body gestures that involve at least one other body part.

From a technical perspective, the categories of this taxonomy could further be split up, as it is done within the FUBI framework in Section 5.4. Therein, I, e.g. additionally distinguish between gestures involving the orientation or position of a joint. Regarding the form dimension, dynamic gestures are further differentiated dependent on whether they only contain a single movement direction (linear or angular) or the movement forms a more complex shape, i.e. a symbol. However, I did not include these additional properties into the taxonomy, as it would make the taxonomy too complex, and some gestures could not be distinguished as they can be described with or actually combine multiple gesture types.

4.2 Gestures for Controlling Humanoid Robots

Recently, researchers are increasingly addressing the use of full body gestures and postures to teleoperate and guide robots and hence enhance the user's natural experience and engagement with the robot (cf. Section 2.7.4). The key to their approaches is to define intuitive and natural human-robot interaction using non-verbal communications, such as body gestures. Again, most of the algorithms that use body gestures to control robots are based on gesture design paradigms that are defined by the developers. However,

as the user is not involved in the process, the designed gestures may not be the most intuitive and may not represent their natural behavior. In order to support the control of robots using intuitive full body interaction, we need to collect data on the basis of the users' body behavior.

In this section, I present how we created and analyzed a set of user-defined body gestures to navigate a humanoid robot. For creating the gesture set, we collect data from both Technical[1] (T) and Non-Technical (NT) users when performing gesture motions to navigate a humanoid robot (Nao). We further suggest implications for humanoid robot control using human gestures. This work was originally published in [135].

4.2.1 Navigational Control of Humanoid Robots

Usually, navigational control of a humanoid robot is done using traditional input computer devices, such as a keyboard and mouse [198, 86] or a joystick [162]. However, the fact that humanoid robots are machines that look like humans and preserve some human functionalities has motivated researchers to look for intuitive interaction ways that are similar to human-human communication as presented in Section 2.7.4.

Previous work in this field relied on the developers of the system to define commands and gestural instructions while approaches that follow a user-centered design approach are rare. An example includes the work by Barattini et al. [9] who defined a gesture set for the control of industrial collaborative robots based on user-centered design criteria, such as physical and mental effort. Ende et al. [45] as well as Gleeson et al. [57] defined gesture sets for robot control based on observations of human-human collaboration. The underlying assumption is that gestures inspired by human-human interaction are easy to remember and to perform.

An approach to gesture design similar to our own approach has been presented by Bodiroža et al. [17]. They conducted an experiment in which they asked users to perform gestures they associated with a given task that was described with verb-noun keywords, such as "bring check". While the approach served to identify appropriate gestures for human-robot control,

[1] We term a user that is experienced with robots and/or gesture tracking as *Technical*

the resulting gestures have not yet been evaluated in such a scenario. Our approach distinguishes from their work by presenting users with videos of robots performing a task as opposed to describing the task verbally. The advantage of our experimental setting for acquiring gestures is the greater similarity that it bears to the setting in which the gestures will be eventually employed. Furthermore, study participants can recognize more easily what the robot should actually do.

4.2.2 User Study

The main objective of the study, described in this section, was to derive a set of body gestures from users spontaneously instructing a humanoid robot. In particular, we focused on navigational control of the humanoid robot Nao and used the eleven actions *Move forward, Move backward, Move left, Move right, Turn left, Turn right, Stop movement, Speed up, Slow down, Stand up and Sit down*, for which the study participants chose gestures.

The motions of those navigational actions were defined from the perspective of the robot and were implemented using the built in motion module of the Nao system (Academic Edition V3.2). We teleoperated the robot through a WiFi connection by implementing several python scripts that used the native API delivered by Aldebaran Robotics. We adopted the Wizard-of-Oz technique to teleoperate the robot throughout each session.

In addition to creating a user-defined gesture set, we wanted to see whether users with a technical background, i.e. with a better understanding of gesture recognition hardware, such as the Kinect, and knowledge about robots and their abilities, used different gestures than participants without a technical background.

Participants

We considered two types of user groups, Technical (T) and Non-Technical (NT): The first were users that had some experience with humanoid robots and were aware of gesture tracking systems (such as the Kinect). The second were users that did not have much experience with any of those

technologies. We considered the two groups as it was apparent when a user was aware of the limitation of the technologies they could define their gestures based on those limitations; hence, including the two groups (T and NT) should allow system designers to consider the both characteristics.

We elicited performed gestural actions from 35 participants (17 T, 18 NT), all from Germany. Initially, we asked participants, on a 5-point Likert scale (ranging from one to five), about their experience with the Kinect and with a humanoid robot. The 17 T participants (6 female, 11 male) had an average experience with Kinect = 2.71 and with a humanoid robot = 2.41. The 17 T participants had an average age of 29 (SD = 5.2) and were mainly from the Computer Science background. While the 18 NT participants (10 female, 8 male) had an average experience with Kinect = 1.11 and with a humanoid robot = 1.06. Most of the 18 NT participants were students from several disciplines, such as the social sciences, linguistics or economics, and had an average age of 27 (SD = 7.8). All participants except one were right-handed.

Setup and Procedure

The experiment was arranged in a room that is 3 meters wide and 6.5 meters deep. The room was equipped with a 50 inch plasma display and two cameras. The first camera recorded the front view of the user, while the other camera was setup as a side camera. The participant had a designated region that he/she was allowed to freely move in during the study. This region was defined from the user's initial position and a distance of about 1 meter around that point. The humanoid robot was placed 2 meters away from the user and is facing them. The setup is depicted in Figure 4.1.

Figure 4.1: Setup for controlling a robot with body gestures

At the beginning of the experiment, each participant was given a description of the study and

was told to stay within their designated region in the room. Each participant was asked to perform the following steps:

1. On the screen, watch a video that demonstrates how Nao performs one of the navigational actions.
2. Upon the completion of the video, perform a gesture that can command Nao to repeat the demonstrated action.
3. Watch Nao performing the corresponding action (this is remotely activated by an instructor).
4. Answer a questionnaire corresponding to the action.

The eleven actions were presented to each participant in a randomized order. For the actions *Speed up, Slow down* and *Stop movement*, Nao was in motion when the gesture was to be preformed by the participant. In this case, participants are asked to state when they are ready, after watching the video on the screen, and Nao was immediately activated then. Subjective and objective measures are explained further in the following section.

4.2.3 Results and Discussion

The results of our study consist of two user-defined gesture sets, the corresponding taxonomy distributions, performance data measures, qualitative observations, and subjective responses.

Gesture Taxonomy

We manually classified all proposed gestures according to the taxonomy described in 4.1.2. In addition, we added a fourth dimension called *view-point* described in Table 4.2.

The view-point dimension is a result of the human-robot interaction scenario. It can be explained best with pointing gestures in a scenario where the robot is facing the user. Thus, a user-centric view-point means that when the user is pointing to his or her right, the robot should move in the pointing direction and, therefore, to the left from the robot's view. The opposite is a robot-centric view-point, i.e. when the user is pointing to his

Table 4.2: Additional dimension for the taxonomy of full body gestures

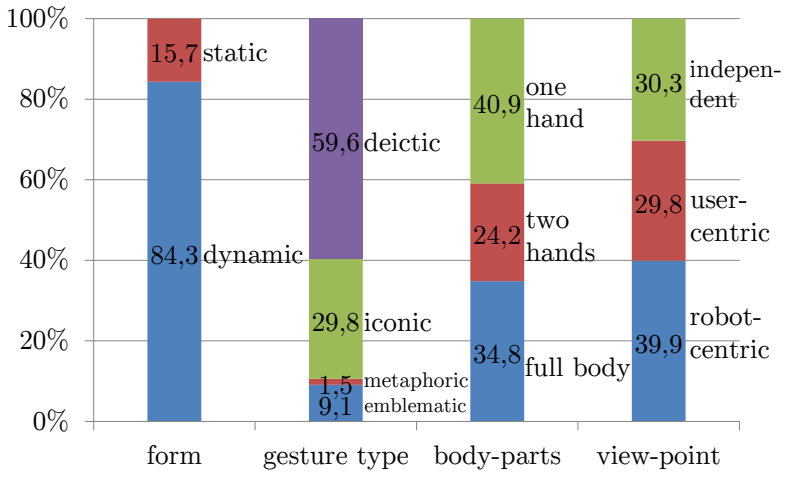
View-Point	independent	The gesture is view-point independent.
	user-centric	The gesture is performed from the user's point of view.
	robot-centric	The gesture is performed from the robot's point of view.

or her right, the robot moves in opposite to the pointing direction (to the right from the robot's view). Other gestures are view-point independent, e.g. an open front-facing hand for stopping which does not include any directional information.

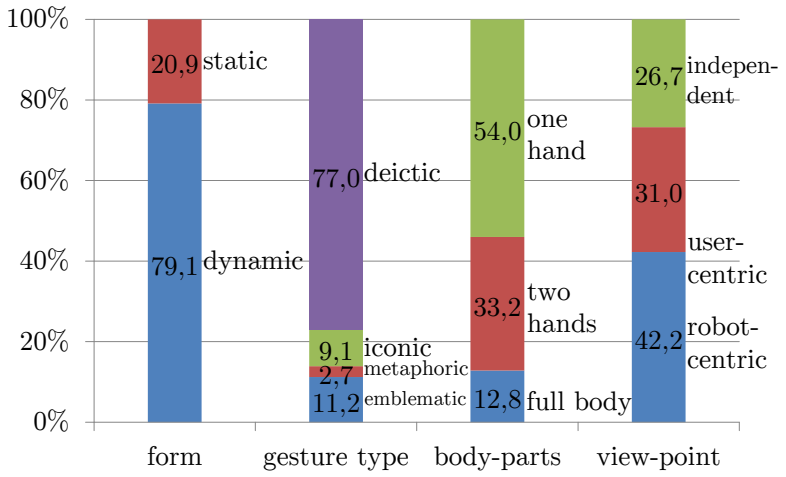
Figure 4.2 depicts the taxonomy distributions for T and NT users. The two most visible differences between the two kinds of users can be seen in the *gesture type* dimension (χ^2 (3) = 26.23, $p < 0.001$) and the *involved body parts* dimension (χ^2 (2) = 25.46, $p < 0.001$). While T users clearly preferred deictic gestures and mainly used their hands for gesturing, NT users more often used full body and iconic gestures. Therefore, one can say that T users preferred more abstract and less exhausting gestures. This was emphasized by the fact that the T users also tended to use more static postures than the NT, however, we found no significant differences for the *form* dimension (χ^2 (1) = 1.75, $p = 0.186$).

Gesture Set

The gestural data collected from the participants of the study was used to define a set of user-defined gestures for the specified control actions. The process of selecting a suitable gesture candidate for a control action was the one described in Section 2.5.1. As we already got multiple first candidate sets $c_1(a)$ with equal sizes for several actions, we did not consider smaller candidate sets $c_{2..n}(a)$. Figure 4.3 depicts the gesture candidates for the eleven actions for both T and NT users. In all of the cases, in which we got multiple first candidate sets, two gesture candidates are present for the action, as there were always two first candidate sets $c_1(a)$ with an equal number of identical gestures, e.g. action "Move forward" for NT.



(a) Taxonomy for non-technical users

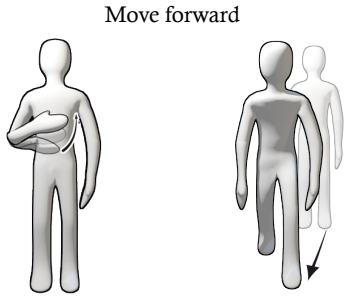


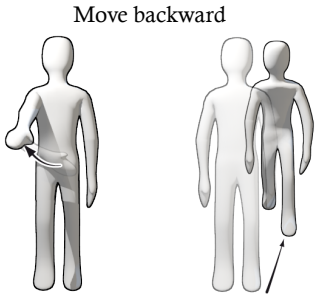
(b) Taxonomy for technical users

Figure 4.2: Taxonomy distributions for user-defined gestures to control a humanoid robot

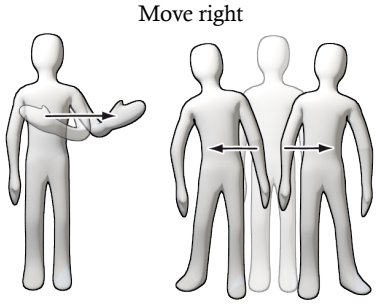
Timings

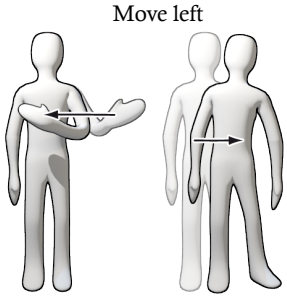
The video recordings of all participants, from the camera videotaping the frontal view of the user, were annotated using the ELAN annotation tools [187]. The annotations segmented each video into eleven actions and each action into four phases (Start-up, Preparation, Stroke, and Retraction). The start-up phase represented the time it took the participants to start their gestural instruction, after watching the action on the screen. The





	T	NT (a)	NT (b)	T	NT
Startup	2.89, 1.71	2.12, 1.52	2.04, 0.65	2.26, 0.91	2.19, 1.57
Preparation	0.29, 0.05	0.15, 0.30	0.54, 0.34	0.40, 0.23	0.27, 0.54
Stroke	1.84, 1.21	4.84, 1.87	2.02, 1.37	1.61, 0.47	4.22, 2.18
Retraction	0.80, 0.44	0.72, 1.44	0.63, 0.17	1.12, 0.79	N/A



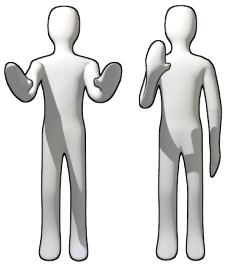


	T	NT (a)	NT (b)	T	NT
Startup	3.00, 1.43	1.00, 0.52	2.39, 0.69	1.78, 0.60	1.44, 0.62
Preparation	0.42, 0.39	0.17, 0.33	0.24, 0.28	0.70, 0.62	0.09, 0.19
Stroke	2.04, 1.91	5.10, 1.65	2.85, 1.38	2.37, 2.34	4.59, 1.54
Retraction	0.84, 0.53	N/A	0.50, 1.01	0.80, 0.24	N/A

	T	NT	T	NT
Startup	2.59, 1.43	2.32, 2.18	2.48, 1.10	1.10, 0.48
Preparation	0.59, 0.57	0.73, 0.76	0.43, 0.18	0.45, 0.21
Stroke	2.28, 1.41	3.91, 2.45	2.19, 0.97	2.83, 1.60
Retraction	1.36, 0.86	0.70, 0.69	0.75, 0.40	0.73, 0.31

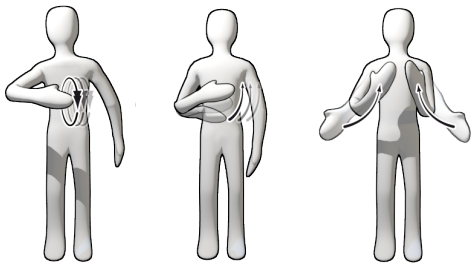
Figure 4.3: User-defined gestures for technical (T) and non-technical (NT) participants to navigate a humanoid robot with the gesture timings (mean and SD in seconds).

Stop movement



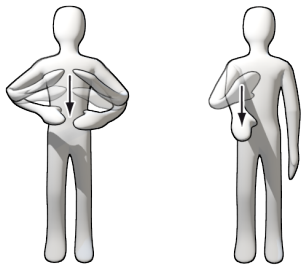
	T	NT
Startup	2.76, 1.26	2.48, 1.26
Preparation	0.32, 0.12	0.24, 0.13
Stroke	2.14, 0.46	2.52, 0.92
Retraction	0.57, 0.25	1.09, 1.09

Speed up



	T (a)	T (b)	NT
Startup	2.07, 1.54	2.27, 1.04	2.67, 1.36
Preparation	0.24, 0.07	0.25, 0.05	0.28, 0.13
Stroke	3.05, 1.51	1.60, 0.11	2.86, 2.03
Retraction	0.74, 0.44	0.63, 0.10	0.95, 0.28

Slow down

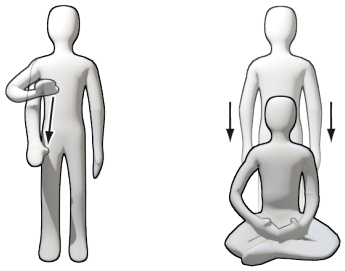


	T	NT (a)	NT (b)
Startup	3.12, 2.36	2.81, 1.17	2.92, 2.24
Preparation	0.37, 0.31	1.06, 1.46	0.89, 1.50
Stroke	3.72, 1.76	3.09, 1.23	2.56, 1.73
Retraction	1.25, 0.83	1.16, 0.47	1.35, 0.80

Stand up

	T	NT
Startup	2.71, 0.92	1.93, 0.94
Preparation	0.57, 0.38	0.72, 0.72
Stroke	2.00, 1.21	3.67, 1.92
Retraction	0.95, 0.22	0.38, 0.38

Sit down

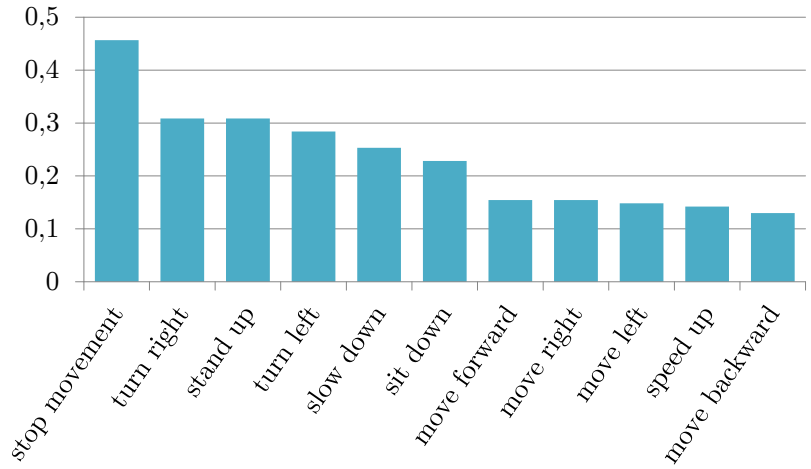


	T	NT (a)	NT (b)
Startup	1.69, 0.65	1.35, 0.27	2.46, 1.58
Preparation	0.37, 0.18	0.35, 0.11	0.99, 0.66
Stroke	2.11, 2.29	1.81, 1.05	6.92, 2.32
Retraction	0.76, 0.21	0.58, 0.12	1.52, 2.08

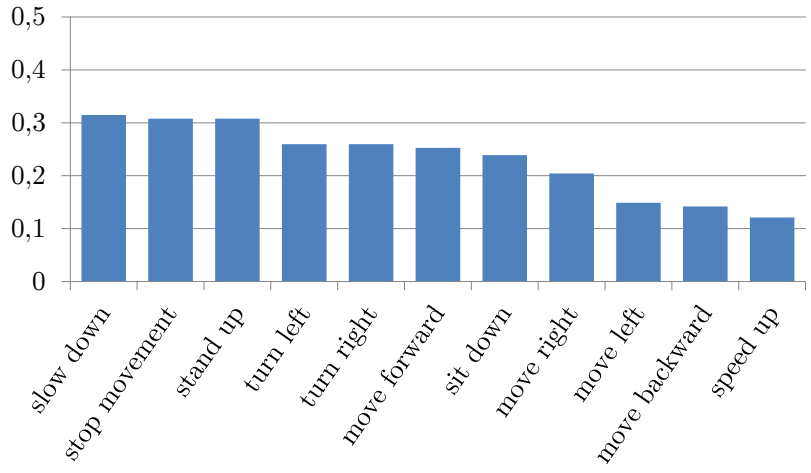
Figure 4.3: Continued

other three values were the times for the gestural phases defined by McNeill [122]. Using the annotation tool, the times for the four gestural phases were extracted for the eleven actions of each participant. Figure 4.3 includes the average times (for T and NT) for each of the gesture candidates' phases.

Agreement Scores



(a) Agreement scores for non-technical users



(b) Agreement scores for technical users

Figure 4.4: Agreement scores per action for controlling a humanoid robot

The agreement scores for the eleven control actions of our study are presented in Figure 4.4. The mean agreement scores for the T and NT

participants were the same with $\overline{AS}$ = 0.23.

User Ratings

After each action, participants were asked to rate the *goodness* and *easiness* of their performed gesture on 7-point Likert scales. The results reveal that the answers for the two mentioned questions correlated significantly for the T group (r = 0.54, $p < 0.01$) as well as for the NT group (r = 0.40, $p < 0.01$). As expected, gestures that were considered as good matches for an action were usually easy to think of and to produce. Beside the direct correlation between *goodness* and *easiness*, we also checked for their correlation with the agreement scores and the timings (especially the *Start Up* and *Stroke* phase), but nothing significant could be found.

Discussion

Most user-defined gestures we found for the navigational control of a humanoid robot were deictic gestures, which indicate a position or direction. Therefore, the main focus of the gesture recognition should lay on this type of gestures. However, we noticed that the gesture view-point may vary especially in these cases. This may pose a great challenge for the gesture recognition: if mirrored gestures should be allowed, how does the robot know if it should move to the left-hand or right-hand side, when the user is pointing to his or her right? A solution could be to offer different modes for the navigational control: one in robot-view and one in user-view. Nevertheless, the interaction designer should think carefully of which gestures would be influenced by the control mode. For example, gestures for linear movements should usually all be influenced depending on the chosen view-point, while gestures for rotating the robot should remain the same. Another interesting point is that one-hand gestures were still the most important ones, however, two-hand gestures were also used relatively often, and NT users performed a considerable number of gestures that involve other body parts. The usage of the second hand mostly resulted in symmetrical gestures, for which the information from the second hand was, more or less, redundant, but could be used to increase the confidence of a recognition system. The

use of full body gestures raised a different issue: they could only be included when implementing additional gesture recognizers, and in opposite to the hand gestures, they really would need the full body tracking information which justifies the usage of a depth sensor with corresponding tracking technology. Users generally preformed dynamic gestures, therefore, simple posture recognition would often be not enough. Moreover, the usual statically labeled pointing gesture should not be optimized for a certain amount of dwell-time as a lot of users included a single or repeated waving motion into pointing to indicate direction.

4.2.4 Conclusion and Future Work

To define the users' preferences in navigating a humanoid robot using gestural commands, we conducted a study on 35 participants that belong to two groups: technology aware users (i.e. gesture recognition and robots), and non-experienced users. The analysis of the data revealed (1) a user-defined gesture set to control a humanoid robot, (2) a taxonomy of the human-robot navigational gestures, (3) user agreement scores for each of the gestures representing a navigational commands, (4) time performances of the gesture motions, and (5) design implications for gesture recognition.

Based on the results of the study, I will present and evaluate an implementation of the user-defined gestures in Section 7.3.

In the study presented in this section, we focused on navigational commands, however, a humanoid robot can do more functions that can be also investigated in future work. In addition, the subjective study revealed that a combination between gesture and speech commands is important and should be further investigated.

4.3 Gestures for Intercultural Training with Traveller

To support experiential learning in simulation environments, technologies are required that allow for intuitive and natural forms of interaction. To

provide basic interaction, users need to be able to move within the virtual environment (navigation) and communicate with virtual characters (dialogue). Traditionally, navigation and dialogue have been controlled with keyboard, mouse or joystick input often accompanied by a graphical interface. This makes the application easier to develop and distribute, as no special hardware requirements have to be met. However, traditional keyboard- and mouse-based interaction modalities do not allow for human-like interaction styles. Furthermore, they do not involve users bodily, which may affect their experience. Better options for the interaction in intercultural learning environments are speech or gesture input that emulate human-human conversation in a more direct manner (cf. Section 2.7.2).

In correspondence to the topic of this dissertation, I focus on full body gestures that users need to perform for triggering the two types of interaction. For navigation, this might be a quite straight-forward choice, as body movements are also used for navigation in real-life. For dialogue the more natural interaction modality might be speech and it therefore could seem a bit awkward to use body gestures for it (cf. Section 3.2). However, conversational gestures are used in real-life for emphasizing or enhancing speech utterances, and sometimes even to replace them, e.g. when performing a head nod instead of saying "yes".

Designing easy-to-use gesture-based interfaces can be a challenge as well. While there is empirical evidence that bodily interaction contributes to a greater sense of presence, some studies also revealed usability issues that negatively affected user experience. For example, Dow et al. [41] compared two versions of the storytelling system Façade: the original desktop version with a 3D cartoon-like interface and typed natural language input using a keyboard, and an Augmented Reality version, where the user walked through a physical creation of the story telling environment wearing a see-through head-mounted display and using speech input in a Wizard-of-Oz design. Their study revealed that interaction in Augmented Reality contributed to an enhanced sense of presence. However, the increased immersion also interfered with the player's engagement. Aylett et al. [6] found that users enjoyed interaction using a dance pad and a Nintendo Wii Re-

mote. However, they also realized that the interaction hampered the users' activities and demanded considerable effort and concentration. For example, the users found it hard to remember the gestures to be executed with the WiiMote, which negatively affected the interaction flow.

For these reasons, it is important that gesture based interfaces have a robust recognition system and the input gestures should be designed in a way that they represent the corresponding in-game actions intuitively. The gestures should be easy to remember and they should not be physically too tiring or difficult to perform in general. Therefore, we also apply the method by Wobbrock et al. [189] for creating a user-defined gesture set of full body gestures in the interactive intercultural training scenario Traveller (Train for virtually every locality). Arbitrary interactive storytelling applications can include a huge variety of specific actions for navigation and dialogue. It therefore seems quite impossible to find a generic set of actions for those two types, and we instead investigate the action set created for our specific scenario. Nevertheless, it should represent a combination of actions typical to interactive storytelling scenarios and we have the hope that our findings also apply to other scenarios without major differences.

In the following sections, I describe the process for creating the user-defined gesture set based on a user study as previously published in [93]. The implementation of the user-defined gestures in FUBI and the further development of Traveller with the integration of an additional graphical interface (cf. [95]) will be presented in Section 7.4. The integration of the gestures in the application with helping mechanisms for the interaction and the evaluation of the complete interaction system will be presented in Section 7.4.1. Apart from that, we published various other aspects of Traveller in [96, 92, 119, 36, 37].

In the next sections, I will first describe the scenario of our application and its intended user interaction. I further explain our interaction study in Section 4.3.3 and present the results of its analysis in Section 4.3.4, which is followed by a short conclusion and future work.

4.3.1 Interactive Storytelling Scenario

Traveller aims to provide intercultural training for young adults (18-25 year olds). It was developed for the eCute project[2]. The users learn by participating actively in the narrative in which they have to interact with virtual characters from different cultures. However, the characters do not represent real cultures, but synthetic ones as defined by Hofstede [69]. The users adopt the role of a character that has not traveled too much for most of his life. The scenario starts at the café of the character's grandmother, in which he receives a letter from his deceased grandfather. In this letter, the grandfather, who liked to travel the world, promises the grandson a "lost treasure" that he should find in a journey through different countries. In each country the grandson has to interact with locals in so-called critical incidents to progress. To be successful, the users have to select the correct interaction options depending on the agents' simulated synthetic culture. In the final country the users will find out that the promised treasure is the experience that the grandfather had while travelling. An overview of all CIs in Traveller is given in Table 4.3.

Table 4.3: Overview of critical incidents (CIs) users face during Traveller

CI	Country	User Task
1	Malahide	get directions from strangers in a bar
2	Malahide	find the park supervisor in a museum and talk to him in order to receive entry permission to a park
3	Malahide	support or blame the supervisor as he knocks over an artefact
4	Demalempire	interact with a train conductor because you have a wrong ticket
5	Demalempire	help out at a café to earn some money
6	Volcano Island	help somebody in need or continue the treasure hunt
7	Volcano Island	interact with people at a party

[2]funded by the European Commission under grant agreement eCUTE (FP7-ICT-257666), http://ecute.eu (accessed 2015-9-15)

The scenario is implemented using the Unity3D game engine and a modified version of the FAtiMA agent architecture for culturally adaptable behaviors of virtual agents [39].

4.3.2 Gestural Interaction in Traveller

Using full body interaction provides an interesting solution for navigating in the virtual scene and, to some degree, also communicating with virtual characters. By default, actions in Traveller are taken by performing a corresponding full body gesture. Body movements are a very natural form of interaction for navigational actions. As conversational gestures are used in real-life for emphasizing or enhancing speech utterances, and sometimes even to replace them, e.g. performing a head nod instead of saying "yes", using gestures also provides a limited, but still natural way for users to communicate in unknown cultures. Figure 4.5 depicts a user performing a formal greeting represented by a bow gesture in the CI 2 (left-hand image image) and performing an informal greeting represented by a waving gesture in the CI 6 (right-hand image).

Figure 4.5: User performing different greeting gestures

We applied the process for creating a user-defined gestures for the introduction in the café and the first two critical incidents. The users' first task was to find out the way to his hotel by interacting with people in a bar (first critical incident, cf. Figure 4.6 left-hand side). In the subsequent

incident users had to find the responsible supervisor in a nearby museum in order to receive entry permission for a park (second critical incident, cf. Figure 4.6 right-hand side). The scene in the grandmother's café and the mentioned two critical incidents together included the following in-game actions to be triggered by the users: yes, no, sit at bar and wait, approach group, ask for directions, leave the bar, ask about supervisor, ask guard to talk to supervisor, approach supervisor, ask permission.

Figure 4.6: Virtual environment of the two investigated critical incidents (CIs) in Traveller

4.3.3 User Study

The experiment was arranged in a room of about 3 meters width and 6.5 meters depth. The participants were standing at a distance of about 2.5 meters in front of a 50 inch plasma display. A camera was placed in a height of about 1.5 meters left of the display to record the users' front from a slightly tilted view. The participants were told that they should place themselves at the initial position, but that they were still allowed to freely move within the camera's field of view during the study. The experimenter was sitting to the left of the participant and controlling the application running on the display via mouse and keyboard.

After a short introduction and a demographic questionnaire that also included a question about the users' experience with body gesture based interaction, the experimenter explained the participants their role in the study. The experimenter ran through the story script of the application

and as soon as a user input would have been requested by the application, text boxes with the currently available in-game actions were displayed as overlays on the virtual scene as depicted in Figure 4.6. At this point, the participants' task was to invent and perform a gesture for each displayed action, one after the other. The participants were told that they were allowed to use their full body for gesturing, but that the gesture itself should mainly be intuitive for them to trigger the requested action. It should, however, have a semantic relation to the action and not consist of simply pointing towards the action label on screen. To keep the process as reproducible as possible, the experimenter always spoke out the action that the user should investigate next and also gave a short explanation about the meaning of the action to avoid misunderstandings. After performing their invented gesture, the participants should indicate on a 7-point Likert scale how easy it was for them to come up with that gesture.

22 participants took part in the study including 4 females. Their age ranged from 22 to 35 with an average of 26.23 (SD = 3.80). All except for one were right-handed. The participants were recruited from our university campus and all had a computer science background. They stated themselves a medium experience with body gesture based interaction of 2.18 (SD = 0.85) on a scale from 0 (no experience) to 4 (practically daily usage).

4.3.4 Results and Discussion

The next paragraphs depict the results of our study, including our gesture taxonomy, a description of the gesture set, user ratings, agreement scores and time performances.

Gesture Taxonomy

The recorded videos were analyzed and annotated using the ELAN annotation tools [187] to extract the stroke phases of all gestures performed by the study participants for each in-game action. We manually classified all performed gestures according to the taxonomy described in 4.1.2. Figure 4.7 displays the overall taxonomy distribution for the 251 performed gestures.

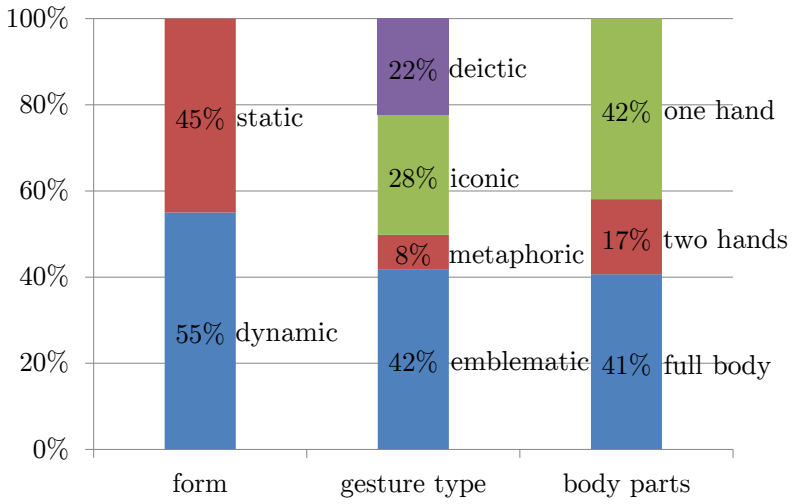


Figure 4.7: Taxonomy distribution for user-defined gestures in Traveller

The frequency of static and dynamic gestures was quite similar. Users tended to perform few metaphoric gesture, a reasonable amount of deictic and iconic gestures, and mostly emblematic ones. They only seldom chose two hand gestures, but roughly an equal number of one hand and full body gestures. The gestures we categorized as iconic according to McNeill [122] in fact were very concrete, which means that most of them were directly miming the meant in-game action, e.g. approach group was often expressed by actually walking a step forward.

Gesture Set

We selected suitable gesture candidates for each in-game action as described in Section 2.5.1 with the adaptions described at the beginning of this chapter. We did not get multiple first candidates $c_1(a)$, but we looked at second candidates $c_2(a)$ in case their size was at least half the size of the first candidate to get more options for the implementation.

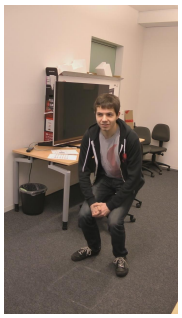
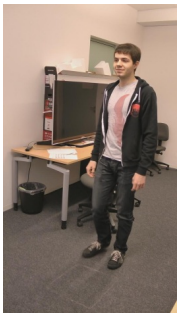
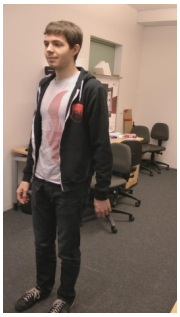
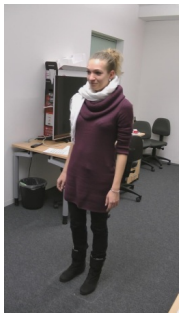
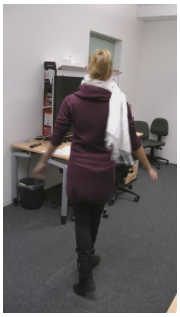
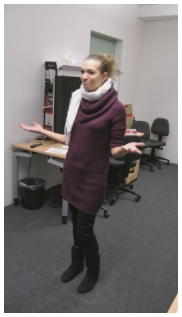
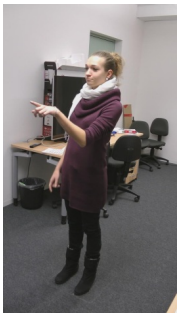
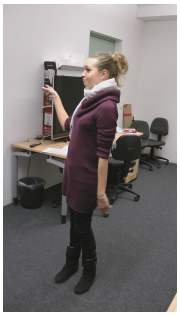
Table 4.4 summarizes the gesture candidates for all ten in-game actions. The third column includes the percentage of how often this candidate was performed among all gestures proposed for this action, and the last three columns depict the candidate's taxonomy. A second candidate was taken into account only in three cases (leave the bar, ask guard to talk to super-

Table 4.4: Gesture candidates for the actions in Traveller

In-Game Action	Gesture Candidates	Occur-rences	Form	Gesture Type	Body Parts
yes	head nod	6%	dynamic	emblematic	full body
no	head shake	68%	dynamic	emblematic	full body
sit at bar and wait	sit down	56%	static	iconic	full body
approach group	step forward	56%	dynamic	iconic	full body
ask for directions	arms out	34%	static	emblematic	two hands
leave bar	turn away	45%	dynamic	iconic	full body
	step backward	27%	dynamic	iconic	full body
ask about supervisor	arms out	50%	static	emblematic	two hands
ask guard to talk to supervisor	point at one after another	38%	static	deictic	one hand
	point to front	21%	static	deictic	one hand
approach supervisor	step forward	56%	dynamic	iconic	full body
ask permission	arms out	23%	static	emblematic	two hands
	tip on shoulder	19%	dynamic	iconic	one hand

visor, and ask permission). The gesture candidates are further exemplified by images of users performing them in Table 4.5. For the actions yes and no, most users chose a head nod or head shake as gestures. The action sit at bar and wait was in most times represented by actually adopting to a sitting position (= sit down). Similarly, we found gesture candidates that represented the action quite directly for approach group, leave bar, and approach supervisor, in which the users did a step backward or a step forward. For the action leave bar a second gesture candidate was turn away which meant the user actually turned around as if going away. Ask permission was additionally expressed by the gesture tip on shoulder that was chosen

Table 4.5: User images of the gestures candidates in Traveller

  head nod	  head shake	 sit down
  step forward (from left to right) / backward (from right to left)	  turn away	 arms out
  point to front (left image) / to one after the other (both images)	 tip on shoulder	

because the supervisor – that participants should ask for permission to enter a park – stood there with the back to them (cf. Figure 4.6 right-hand side: the virtual character at the back), so they assumed they first needed to get his attention. This is also the reason why we labeled the gesture as iconic, although it does not depict "ask permission" directly, however,

it directly depicts "draw attention". For the ask actions we often got the gesture arms out that always included moving the arms to an outward position with open hands, often accompanied by raising the shoulders. The only action for which the gesture candidates were pointing gestures was ask guard to talk to supervisor. Participants either chose to point in the direction of the supervisor (point to front), or to point at the guard first and only afterwards to the supervisor (point at one after another).

Timings

Table 4.6: Times for one stroke of the gesture candidates in Traveller

in-game action	gesture candidate(s)	mean time	SD	MIN	MAX
yes	head nod	0.822	0.374	0.420	1.560
no	head shake	0.687	0.233	0.474	1.420
sit at bar and wait	sit down	0.661	0.635	0.120	1.970
approach group	step forward	1.508	0.702	0.422	2.770
ask for directions	arms out	0.707	0.392	0.295	1.375
leave bar	turn away	1.738	0.908	0.557	3.400
	step backward	2.041	0.593	1.295	3.000
ask about supervisor	arms out	0.589	0.341	0.200	1.280
ask guard to talk to supervisor	point at one after another	1.013	0.257	0.540	1.410
	point to front	0.560	0.558	0.130	1.625
approach supervisor	step forward	1.444	0.659	0.517	2.470
ask permission	arms out	0.759	0.578	0.190	1.770
	tip on shoulder	0.356	0.066	0.257	0.410

Table 4.6 depicts the times it took the users to perform one stroke of the gesture candidates. One stroke means e.g. for the head nod gesture that the user moves the head from the resting position upwards, then downwards under the resting position, and upwards to the resting position again. In other words, one stroke consists of the minimal gesture that can be found when dividing the gesture into equal sub gestures. For static gestures, one stroke consists only of a hold phase in which the user holds the relevant posture until moving back to a resting position. The table enlists the average times as well as the standard deviation, the minimum, and maximum among the different gesture performances.

Agreement Scores

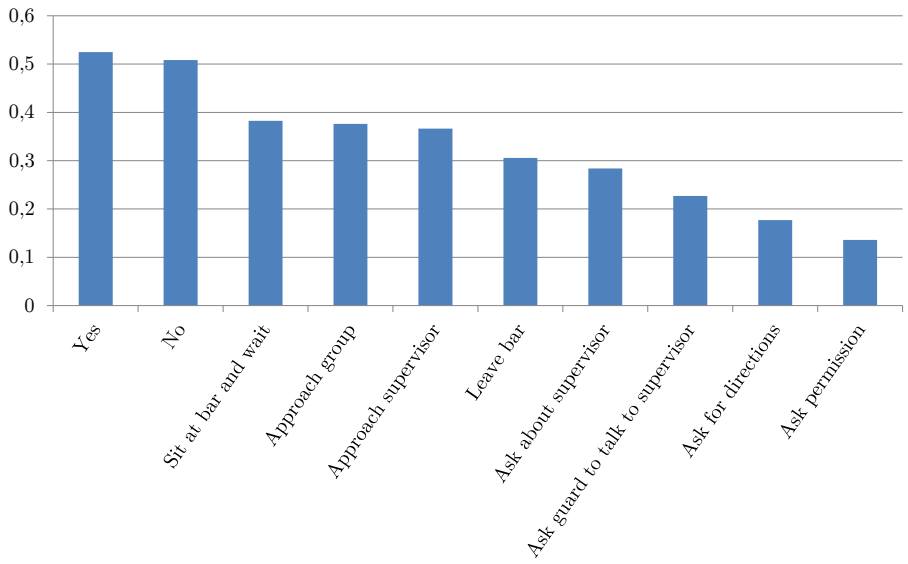


Figure 4.8: Agreement scores for ten actions in Traveller

To further investigate the level of agreement among the participants, we calculated the agreement score as described in Section 2.5.1. The overall agreement for our action set was $\overline{AS}$ = 0.329 (SD = 0.129). Figure 4.8 depicts the agreement scores of the different actions ordered from highest to lowest agreement. Similar to the user ratings, more complex actions (i.e. all "ask ..." actions) caused lower agreement between the users and we

got a large number of different gestures for those. In fact, the results reveal that the level of agreement between the participants strongly correlates to the easiness to invent gestures (Pearson's $r = 0.812$, $p < 0.01$). When the participants thought it was easy to find a gesture for an in-game action, more participants chose the same gestures, and in opposite, when the participants thought it was difficult to invent a gesture for an action, we got a higher variation of gestures as well.

User Ratings

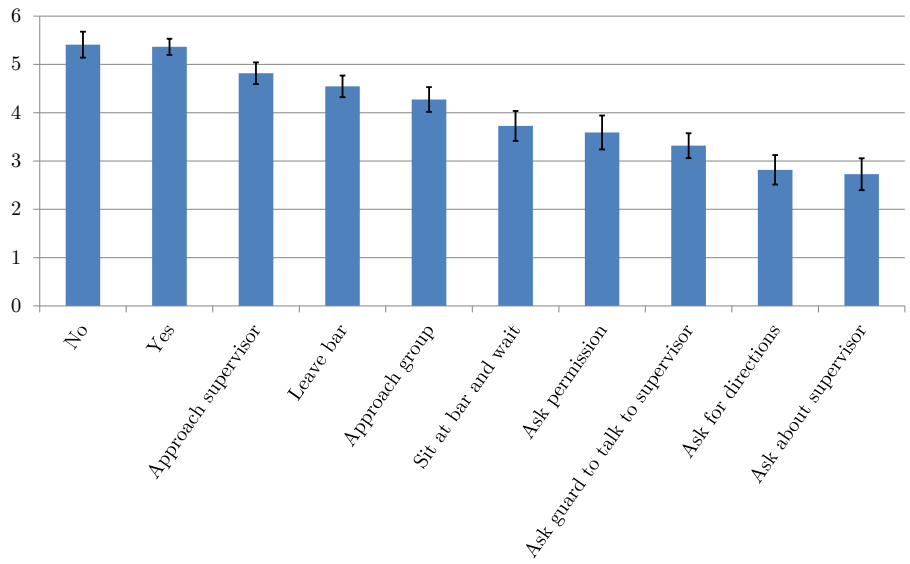


Figure 4.9: User difficulty ratings of ten actions in Traveller

Figure 4.9 depicts the average user ratings for the easiness to invent the gestures for the ten in-game actions on a scale from 0 (very hard) to 6 (very easy). Error bars represent the standard error. The actions are ordered according to their user rating.

A one-way repeated measures ANOVA revealed that the ratings differed significantly between the different actions with $F(9, 21) = 15.90$, $p < 0.01$, $\eta^2 = 0.43$. Post-hoc tests with Bonferroni correction showed that the more complex conversational actions were perceived as more difficult to invent a gesture for them. Accordingly, all actions that include asking character(s)

something were rated as the most difficult ones. In particular, all those actions were rated significantly lower ($p < 0.01$) than the actions yes and no. Approach supervisor and leave bar, were also rated significantly higher ($p < 0.05$) than the ask actions, except for ask permission. Approach group was only significantly higher rated ($p < 0.05$) than ask about supervisor. We found no significant difference between sit at bar and wait and the ask actions, and there was also no significant difference between the ratings of the different ask actions.

Discussion

As far as the taxonomy distribution of our gesture set is concerned, we got quite different results in comparison to our work that investigated user-defined gestures for navigating a humanoid robot (see Section 4.2). We got much less deictic gestures as in the previous work, but more emblematic and also more metaphoric ones, although the latter were still quite rare. This is due to the target of the gestural interaction. While deictic gestures seem to be more suitable for navigational actions, our action set also included conversational actions that need to be described with more abstract gestures due to their increased semantic complexity.

We also had a closer look at the taxonomy distribution of each action itself that revealed that the users never used iconic gestures for the conversational actions (all ask actions plus yes and no) except for the gesture tip on shoulder of the action ask permission. The other actions – which can be categorized as navigational actions – included all types of gestures, and especially a large amount of iconic ones. For further increasing the information content of their gestures, the participants more often used other body parts than their hands in opposite to the robot navigation task. However, they used less dynamic gestures, which indicates that full body gestures often provide enough information in a static version.

As full body gestures were frequently chosen in general, it can be said that this kind of gestures is worth to be used in interactive storytelling scenarios. It seemed that full body gestures are especially intuitive for triggering the in-game actions that occur in this kind of scenarios.

We proposed to select the gesture candidates according to how many users chose a gesture for one in-game action. However, this does not always have to be the best choice. For example, it makes no sense to give the user the choice between two actions represented by the same gesture at the same point in time. In this case it is better to select a less often chosen gesture candidate for at least one of the actions.

There are also other cases in which it is helpful to select a different gesture, e.g. if the recognition software is not able to detect the gesture in a robust way. A more specific reason for doing this is also given in our application. As we aim at intercultural training, we want the users, at a later point in our scenario, to be confronted with gestures that are unfamiliar to them, as this can occur when traveling to different countries. For this purpose it might also be worth to conduct the study with participants of different cultural background to get a different gesture set.

Another challenge we faced was the problem of potentially too complex in-game actions, and especially the difficulty to represent verbal actions with non-verbal gestures. This can be seen in the relatively low agreement scores and user ratings we got for all actions that involve asking virtual characters about something. At the state of the study, our scenario never included multiple ask actions in parallel, so we had no problems with their ambiguous gestures, but in the later development, it was necessary to include an additional kind of interaction as described in Section 7.4.2.

4.3.5 Conclusion

In this section, I presented how we produced a user-defined gesture set for the intercultural training scenario Traveller. During this process we obtained the taxonomy distribution for our interaction set, user ratings and agreement scores for each in-game action, and the time performances of all gesture candidates. An implementation and evaluation of the gestures candidates will be presented in Section 7.4.

Chapter 5

The Full Body Interaction Framework

For investigating full body interaction, I decided to implement a software framework that helps integrating this kind of interaction in various applications. The development started for the application described in Section 3.1 which is based on [98]. During this research, we realized that it would be helpful to have software that can be reused to integrate full body interaction in different applications and which makes it easy to adapt the interaction at a later point in time. The framework is called **FUBI** as an abbreviation for "full body interaction" and its logo is displayed in Figure 5.1.

Figure 5.1: Logo of the full body interaction framework FUBI

The first comprehensive description was given in [97] that is described in parts in Section 5.2. FUBI is freely available under the terms of the Eclipse Public License v. 1.0. A download link is provided on its website

that includes documentation as well: http://hcm-lab.de/fubi.html. By end of August 2015, FUBI has been downloaded more than 3.700 times, with an average of about 100 downloads per month in the last two years. The main development of FUBI was done for the eCute project[1] and its Traveller application described in Section 4.3, however it has been used in multiple other research projects as well, e.g. Tardis[2] or CEEDs[3].

In the next section, I will first describe FUBI's technical architecture. The following three sections will describe FUBI's main functions: avatar control, freehand GUI interaction and full body gesture recognition. For avatar control and freehand GUI interaction, I will directly describe sample applications with corresponding evaluations. For full body gesture recognition, I will give a more detailed technical insight in Section 5.4.

5.1 Technical Architecture

The FUBI framework was developed with six goals in mind:

1. FUBI should support various software and hardware for getting image streams and tracking users within that data. This makes FUBI more universally applicable and it forms the basis that it can stay compatible with new sensor hardware or tracking software.

2. FUBI should offer an easy but powerful way to define or record full body gestures and also help developers testing the gestures. In this way even users without a strong background in gesture recognition or even with few programming skills can develop own gestures and use them in their applications.

3. It should be possible to use the gesture definitions with different hardware and tracking software, so that one does not have to recreate the gestures in case of switching to a new technology.

[1] funded by the European Commission under grant agreement eCUTE (FP7-ICT-257666), http://ecute.eu (accessed 2015-9-15)
[2] funded by the European Comission under grant agreement TARDIS (FP7-ICT-288578), http://www.tardis-project.eu (accessed 2015-9-15)
[3] funded by the European Comission under grant agreement CEEDs (FP7-ICT-258749), http://ceeds-project.eu (accessed 2015-9-15)

4. The gesture recognition techniques should be very efficient, so that multiple gestures can be recognized in parallel in a real-time system. Nevertheless, it should be possible to cover as much as possible kinds of gestures. FUBI should therefore be usable in many different application scenarios.

5. Besides the recognition of gestures and postures, FUBI should also provide avatar control and freehand GUI interaction to be as universal as possible.

6. FUBI should make it easy to integrate the full body interaction in an arbitrary application and it should be possible to reuse gesture definitions for avoiding unnecessary reimplementations.

FUBI supports OpenNI + PrimeSense NiTE (version 1.x and 2.x), the Microsoft Kinect SDK (version 1.x and 2.x), and the LEAP Motion SDK (version 1.x and 2.x) to integrate image streams and tracking data using the corresponding hardware sensors: Microsoft Kinect (for Xbox 360 and One and for Windows version 1 and 2), PrimeSense Carmine, Asus Xtion PRO, and the LEAP Motion Controller. To support various tracking software and hardware, FUBI needs a modular interface for integration.

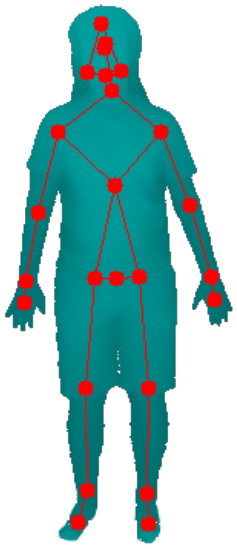


Figure 5.2: FUBI's user skeleton

As not every tracking software offers the same functionalities in the same way, FUBI also needs to convert parts of the data in a common format. This allows to use the same gesture definitions with different tracking hardware and software as well. The most obvious example for data that needs to be converted in a common format is the tracking skeleton. FUBI originally adopted the OpenNI 1.x skeleton which consisted of 15 joints for head, neck, torso, shoulders, elbows, hands, hips, knees and feet. Later, the FUBI skeleton was enhanced with five joints of the Microsoft Kinect SDK tracking for wrists, hands and hip center (which was renamed to waist), and

in addition with the five custom face joints nose, ears, forehead and chin. This forms the FUBI skeleton with 25 joints as displayed in Figure 5.2. For the LEAP Motion Controller, FUBI supports an additional hand skeleton depicted in Figure 5.3 with the six joints palm, thumb, index finger, middle finger, ring finger, and pinky.

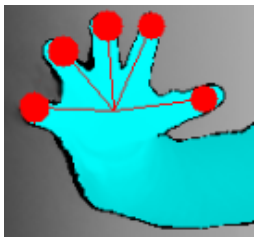


Figure 5.3: FUBI's hand skeleton

The Joint transformations are defined in the FUBI coordinate system, which has its origin at the location of the depth sensor (for LEAP: minus the sensor offset position). The x-axis is pointing to the right-hand side from the view of a user facing the sensor, the y-axis is pointing upwards, and the z-axis is pointing from the sensor to the user (cf. Figure 5.4).

For defining gestures in FUBI, I designed an XML based language, which is described in more detail in Section 5.4. For designing and testing the gestures I developed two sample applications, an OpenGL based one with console output (cf. Figure 5.5) and a WPF based one with a GUI (cf. Figure 5.7). Both samples are included in the FUBI download and should give basic examples for most of FUBI's API methods, but also allow to design and test the gesture definitions.

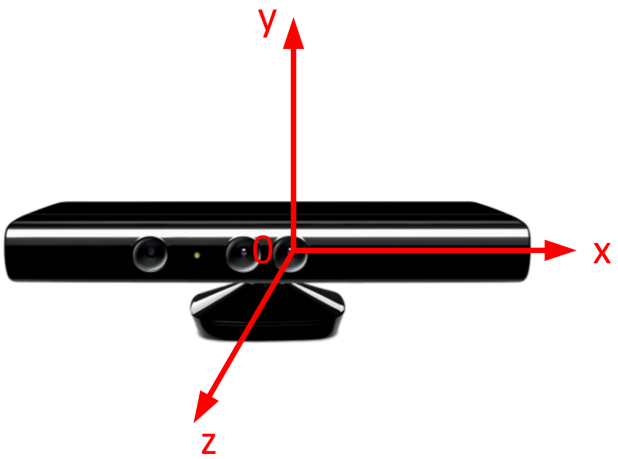


Figure 5.4: FUBI's coordinate system

From a software perspective, the FUBI framework consists of about 30,000 lines of source code (excluding blank lines and comments) that are

structured as follows: The framework's main part is the FUBI project which provides FUBI's C++ API in the *Fubi.h* header, while all main features are organized in the *FubiCore* singleton class. For configuring the FUBI build and, for example, de-/activating certain parts of FUBI and its logging functions, the *FubiConfig.h* can be edited by manipulating the provided preprocessor defines.

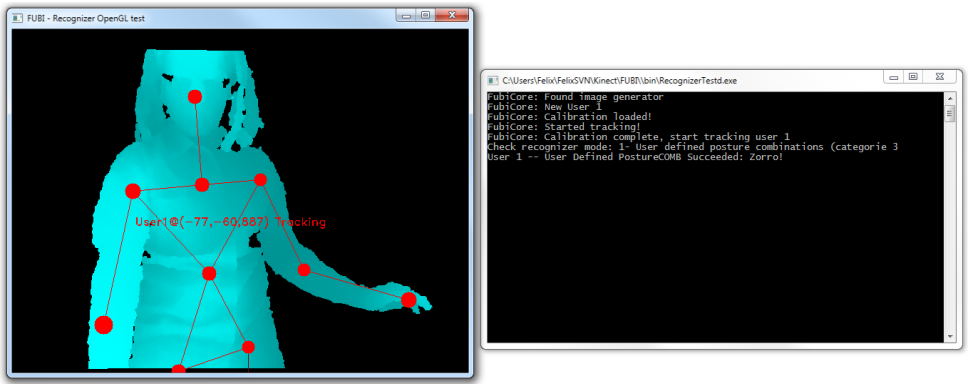
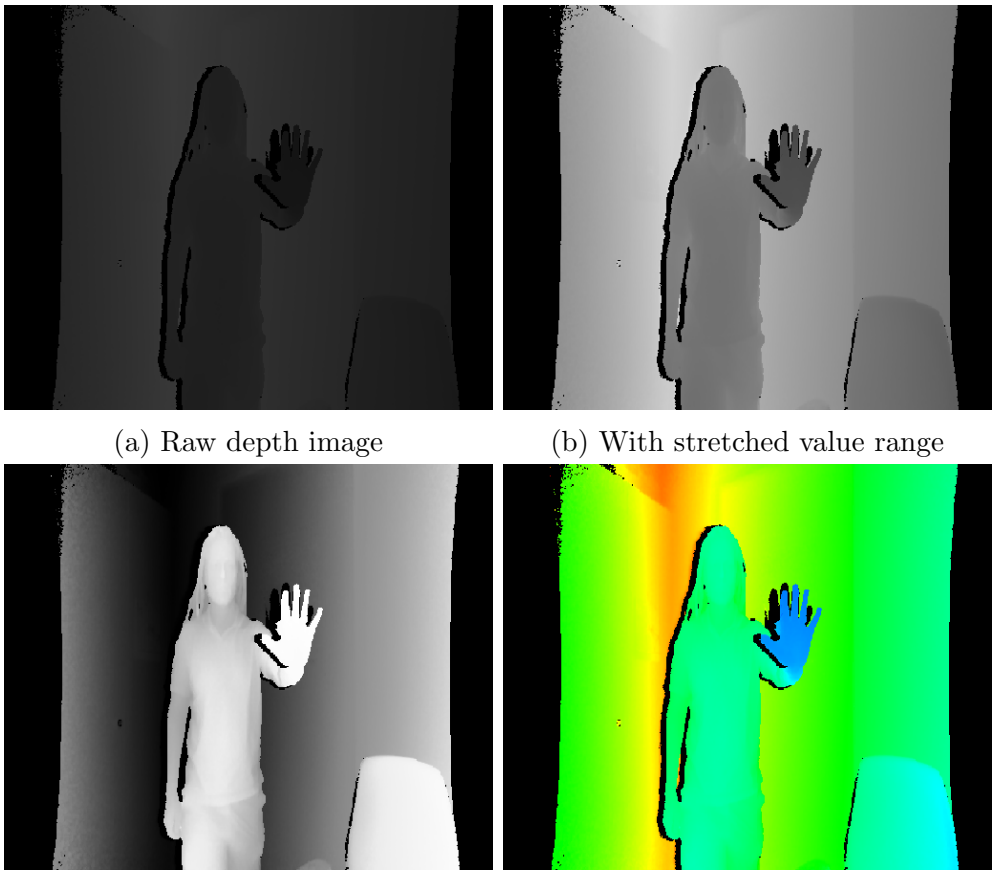


Figure 5.5: FUBI's sample OpenGL application

FUBI further includes integrations of the different supported body tracking software that implement the *FubiISensor* interface and which are: *FubiKinectSDKSensor*, *FubiKinectSDK2Sensor*, *FubiOpenNISensor*, *FubiOpenNI2Sensor*. The *FubiLeapSensor* is another type of sensor which implements the *FubiIFingerSensor* interface as it only provides tracking for finger joints. Besides of the tracked joints, the sensors usually provide a confidence value describing the quality of the tracking.

Using those sensors, the FUBI API provides access to their different image streams, i.e. color, depth and IR images, which can be converted to have one three or four channels and a pixel depth of 8 bit, 16 bit, or 32bit float. Especially mentioned should be the additional modifications for the depth image as depicted in Figure 5.6: one can get access to the raw depth image, however, in this format, the depth information is not very visible to the human eye. Therefore, FUBI also offers modifications of the depth image, i.e. by stretching the value range to the whole range provided by the configured color depth, by amplifying changes between neighboring pixels using a histogram (this is the default for most Kinect SDK / OpenNI

samples), converting a part of the depth information to a color scheme, i.e. the distance is first split into several color ranges from blue=close, over green and yellow to red=far; the brightness of the pixel then determines the distance within the range.



(a) Raw depth image

(b) With stretched value range

(c) Emphasized neighboring pixels by using a histogram

(d) Parts of the depth information converted to color

Figure 5.6: FUBI's provided depth modifications

Optionally, the streams can be enhanced with tracking information. FUBI allows to render the recognized user shapes (which will be colored depending on the user ID, e.g. cyan for user 1, yellow for user 2, ...) and the simplified skeletons (having the complementary color of the user shape) including all (or only a part of the) tracked joints. The latter can further be enhanced by textual information about the joint positions or orientations (local or global; raw or filtered) and calculated body measurements (body/

torso height, shoulder/hip width, upper/lower/complete arm/leg length). In addition, the tracking image can be enhanced by the information provided by (optional) face and finger tracking. All code parts that convert or enhance the image streams and further analyze them to provide finger detection are included in the *FubiImageProcessing* class. Using the tracking capabilities of the different sensors, FUBI holds arrays of all tracked users and hands using its *FubiUser* and *FubiHand* classes. Those include the current state of a user/hand regarding the joint tracking as well as the finger tracking, the calculated body measures, and the progress of the combination recognizers started for that user. Apart from enhancing the image with information, FUBI also allows to crop the image around a special joint of interest, which can be useful for applying additional computer vision algorithms on a specific image part, e.g. face tracking on the head region, or finger tracking on the hand regions of a user.

Regarding the tracking information, FUBI provides several additional options. Besides, the raw tracking positions and orientations coming from the tracking software, it also keeps a filtered version of those transformations. The filtering is done using the adaptive low-pass filter as presented by Casiez et al. [24] which can be configured with three variables: the minimum cutoff frequency defines the filtering applied to a non-moving joint; lower values result in generally smoother data, but higher latency. The cutoff slope further defines how fast the cut off frequency will be increased when a tracking point moves with higher velocity; with lower values, the cut off frequency never changes much and all tracking points get filtered similarly, with higher values, the cut off frequency will adapt faster and you will almost get raw data for tracking points that have a certain speed. An additional cut off frequency configures the filtering for the velocity value that is used for adapting the actual filtering. Altogether, this filter applies a relative strong filtering on joints that are moving slowly for reducing jitter, but it does not filter faster movements much for avoiding delays.

Raw joint positions are in real world space (*mm* values for all three axes). As they are calculated using the depth sensing capabilities of the sensor, the positions are always relative to the infrared camera used to

calculate the depth image. For knowing which pixel position within the depth image corresponds to a real world joint position, FUBI can project the position back to depth space using information provided by the tracking software about the camera setup. In addition, the coordinates can as well be converted to the coordinate space of the infrared or color image, as the tracking software usually provides information on how the different camera spaces are aligned.

The user further has the choice to receive the (real world) transformations in global coordinate space, i.e. position vectors are relative to the sensor position and orientations are defined around the global coordinate axes. Despite, one can also get the transformations in local coordinate space, i.e. position vectors are always relative to the overall body transformation, i.e. the torso position and rotation is removed, and orientations are relative to the orientation of the preceding joint in the skeletal hierarchy, e.g. the elbow orientation is related to the shoulder orientation, which aims to result in how much the elbow joint is physically changed.

The options on how the to render the image streams as well as all other important options, data formats and utility features are include in the *FubiUtils.h*, while the *FubiMath.h* provides classes and functions for mathematical operations, such as vectors, quaternions, matrices with corresponding geometrical operations, but the *FubiUtils.h* also includes other mathematical functions specific to gesture recognition, such as filtering based on Casiez et al. [24], normalizing and resampling partly based on the Dollar $ 1 [190], applying dynamic time warping based on [128], or poly line reduction based on [54]. The *FubiGMR* class and the *FubiGMRUtils.h* further include helping functions for calculating a gaussian mixture model using an expectation maximization algorithm and applying regression on that model. Those latter functionality is based on the software by Calinon [23].

FUBI always keeps single instances of the *FubiRecorder* and *FubiPlayer* classes which – as their names suggest – are used to record and playback skeleton data. *FubiRecorder* records one user or hand at a time and stores the data in an XML format containing positions and orientations for all joints as well as the currently calculated finger counts. The data is organized

in frames which get an id and time code. *FubiPlayer* can load such a file and playback the tracking data on a user or hand with a special ID. The recorded data files are as well used in the later described recognizers for symbolic gestures 5.4.7.

The last, but actually most important part of the FUBI project are the gesture recognition features which are structured as follows. The actual recognition part is placed in the "GestureRecognizer" folder which includes various kinds of recognizer classes. Those first include classes for all basic gesture recognizers that implement the *IGestureRecognizer* interface. The only additional recognizer class are the combination recognizers. Both classes are explained in Section 5.4. The folder further includes several predefined recognizer classes which each implement a certain gesture only. However, they have become obsolete in the meantime, as all of them can be configured using XML as well. While the *FubiPredefinedGestures.h* includes the ids and names of the obsolete predefined gesture recognizers, the *FubiRecognizerFactory.h* supports the instantiation and configuration of any gesture recognizers during runtime. This feature is also used by the *FubiXMLParser* class which provides functions that get an XML file as input and instantiate all valid recognizer configurations included in that file. For this purpose, it makes use of the rapidXML parser by Marcin Kalicinski.

In addition to the C++ API, FUBI further provides a C# wrapper that offers access to all important (C++) API functions. It further holds own (C# versions) of *FubiUtils* and *FubiPredefinedGestures*.

FUBI's WPF based *GUI* organizes its main functionalities within multiple tabs at the top of the main window (cf. Figure 5.7, 1). In the main tab (cf. Figure 5.7, 2), one can switch between all supported tracking software, select the image stream to be used, change the rendering modifications for the depth stream (cf. Figure 5.6), load new recognizers from an XML file, clear all recognizers, enable rendering of finger sensor image streams, and open the recognizer status windows which provides information about the progress of all combination recognizers, e.g. their current state, whether they are in transition, or what the user should do at the moment to keep it going (cf. Figure 5.7, 6). The FUBI GUI further includes the output

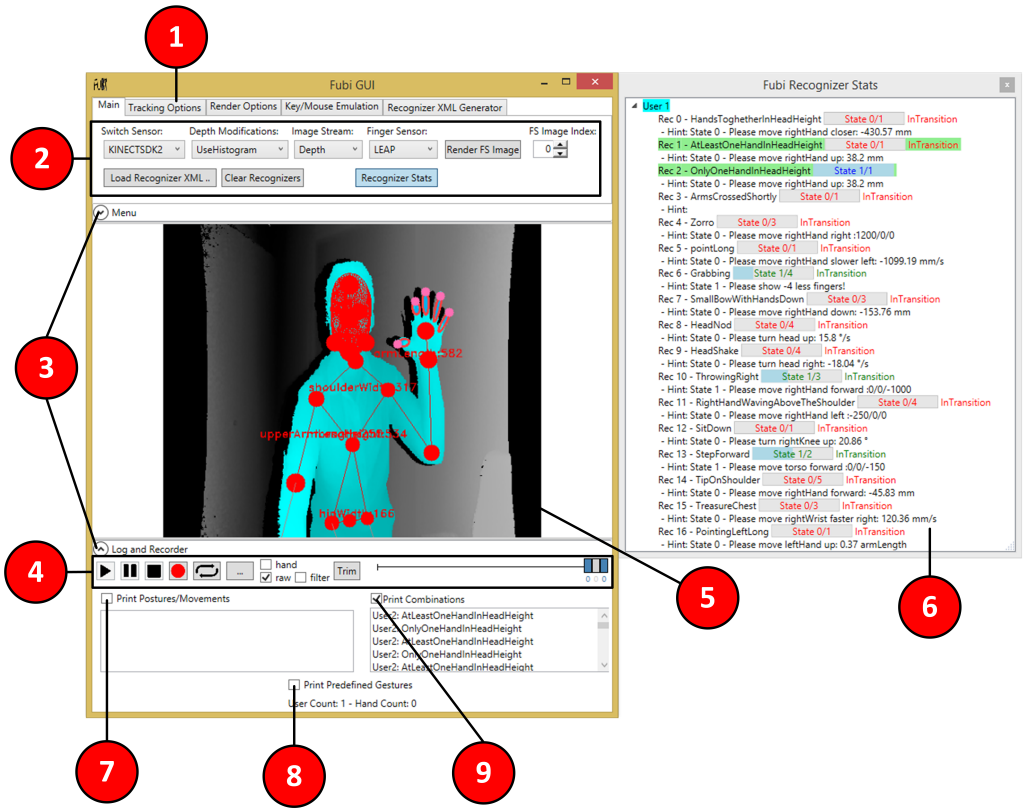
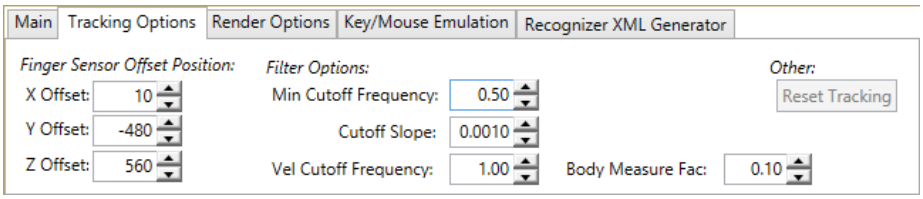


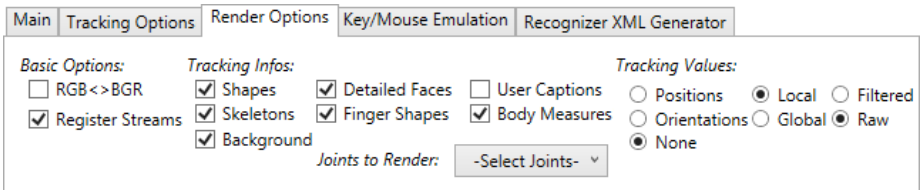
Figure 5.7: FUBI's GUI: 1: tabs with the GUI's main functionalities; 2: main tab; 3: buttons for minimizing the top and bottom part of the GUI; 4: playback and recording controls; 5: output image; 6: status window with information about the progress of combination recognizers; 7–9: checkboxes for enabling/disabling the logging of basic recognizers, combination recognizers, and recognizers predefined in C++-code;

image depending on the currently selected stream and chosen rendering options, e.g. Figure 5.7, 5 displays the depth image with colored user shapes and tracking skeletons including face tracking, body measurements, detailed faces and finger tracking. In the bottom of the main window, the FUBI GUI has several controls for playing and recording skeleton data (cf. Figure 5.7, 4), and two text boxes for logging successful recognitions of the selected recognizer types (cf. Figure 5.7, 7–9).

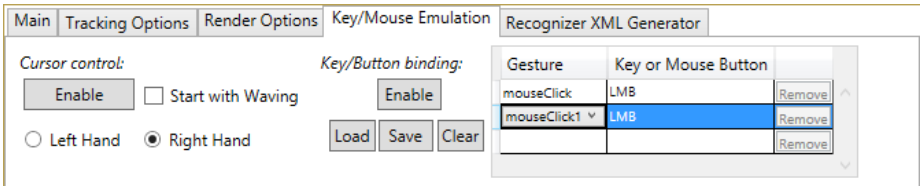
In the second tab (cf. Figure 5.8a), the FUBI GUI has additional tracking options, i.e. the offset position for the finger sensor (e.g. LEAP Motion Controller) that registers the finger sensor to the main sensor (e.g. Kinect)



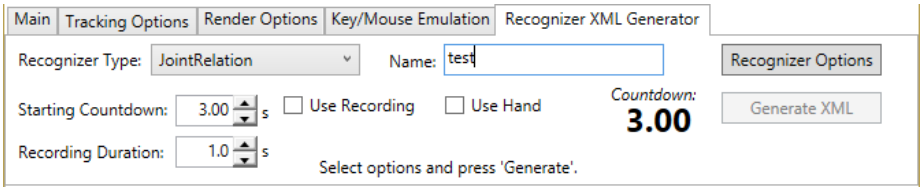
(a) Tracking options



(b) Render options



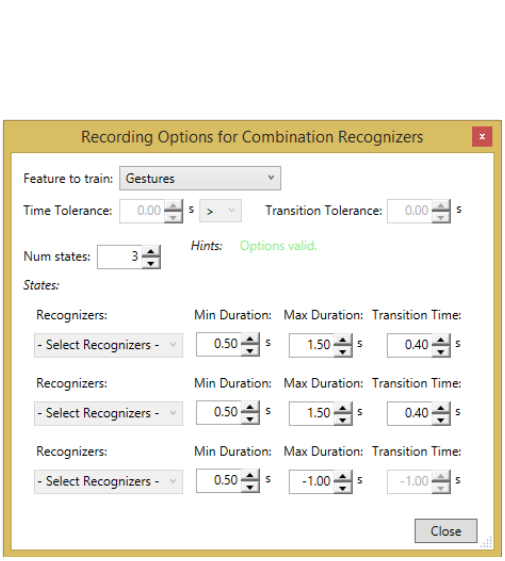
(c) Key and mouse bindings



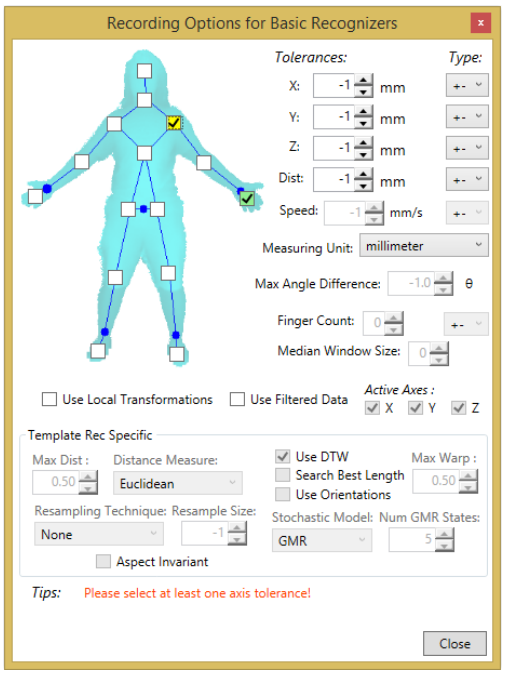
(d) XML generator

Figure 5.8: FUBI's GUI Tabs

and additional filter options for the actual tracking data and the body measurements. The filter options for the tracking data are described later in this section. For the body measurements, a simple low-pass filter is applied, for which the *Body Measure Fac* defines the percentage of how much new data is taken into account in each update of body measures (currently called every 0.5 seconds). The third tab (cf. Figure 5.8b) allows users to select the tracking information that should be rendered onto the output image, and in the subsequent tab (cf. Figure 5.8c), they can enable mouse emulation to control the Windows cursor with one hand. This tab further allows to bind gestures to any key or mouse events for completely controlling arbitrary applications or the Windows desktop with full body interaction.



(a) Combination Options



(b) Basic RecognizerOptions

Figure 5.9: FUBI's GUI XML Generator Options

The last tab (cf. Figure 5.8d) helps the users at creating the XML gesture definitions by training the recognizer's values with an actual gesture performance. Therefore, the user first has to select the type of recognizer that he or she wants to generate and give it a unique name. The actual training can either be done by using a previously recorded gesture performance, or by performing the gesture directly. Both cases can be used with normal user tracking or with a finger tracking sensor such as the LEAP Motion Controller. If the gesture should be performed directly, the user can adjust a count down to define when the training starts and the training duration. There are additional options for configuring the recognizer training of basic recognizer (cf. Figure 5.9a) or combinations (cf. Figure 5.9b).

The different recognizer types are described in Section 5.4, nevertheless, I briefly describe the options for training them in the following. For basic recognizers (cf. Figure 5.9b), you have to select the joint(s) that take(s) part in your gesture. Then you can define tolerance values for each axis (Note: -1 ignores the axis) and their type ($\pm$ for both directions, $<$ or $>$ for

one direction only). Similar for the finger count tolerances, for which you can additionally set the median window size. For linear movements, you can instead define a speed tolerance, the maximum angle difference, and the active axes. Depending on the recognizer type, the units within the GUI change automatically, but in addition, you can change the measuring unit from millimeters to body height, arm length etc. and you can choose to use local or filtered data. For template recognizers, you can – similar to the linear movements – select which axes should be taken into account as well as a maximum angle difference, which here restricts the rotation invariance. You can further select the maximum distance at which an input will be regarded as recognized and the distance measure (*Euclidean*, *Manhattan*, *Malhanobis*, or *TurningAngleDiff*). You can define the resampling technique (*None*, *EquiDistant*, *HermiteSpline*, or *PolyLine*), and a fixed resampling size if required. You can activate or deactivate, whether the gesture should be recognized with any aspect ratio, whether it is defined by orientations instead of positions, whether to apply a GSS search for finding the best window length, and whether to apply DTW. For the latter, *Max Warp* defines how much percentage of the gesture are allowed to warped. Finally, you can select a stochastic model. At the moment you only have the choice between *NONE* and *GMR*, and you can further set the number of states for the *GMR*. You may also want to take a look at the text at the bottom of the window, as it tells you whether there is something wrong with your configuration.

For Combinations (cf. Figure 5.9b), you have four options for the features you want to train: *none*, *gestures*, *times*, or *both*. In all cases, you have to define the number of states that your gesture should contain. If you select *none*, nothing will be recorded, but you have to select the recognizers of each state as well as its time constraints (minimum/maximum duration and transition time) by hand. If you select *gestures*, you only have to define the time constraints, and during the recording, the tool will try to find out which gestures are performed during each state. You should load an XML file including (only) the recognizers that you want to use during your combination gesture in this case. If you select *times*, you have to define

the recognizers for each state, but during the recording, the tool will try to figure out the timings (how long you perform them, how much time is in between them). In this case you can also define tolerance values for the min/max durations (*Time Tolerance*) and for the transitions (*Transition Tolerance*). If you select *both*, you only specify the number of states and the two tolerance values, but the tool tries to find out the gestures and timings for each state during the recording. As the latter has the least preconditions, it also has the highest risk to fail.

We also implemented different applications that integrate full body interaction using FUBI which are described in Chapter 3 and 4. One important type of target applications are in the field of social signal processing. Therefore, we integrated the FUBI framework in the Social Signal Interpretation framework (SSI) [183], in which e.g. certain body postures are recorded for later analyzing higher level features such as the openness of a participant during a conversation. Another important type of target applications are applications with virtual environments ranging from desktop applications to virtual and augmented reality. We therefore integrated FUBI in our own game engine, the Horde3D GameEngine, which uses the C++ API as well as in the Unity3D game engine which uses the C# wrapper. For both exist sample applications as well. As both avatar control and a freehand GUI are very dependent on the used game engine or visualization framework, those features are not implemented in FUBI itself, but they are part of the FUBI integrations for the two mentioned game engines.

In the next two sections I will provide further information on those two types of interaction, while Section 5.4 explains the FUBI's gesture and posture recognition in more detail.

5.2 Full Body Avatar Control

Full body avatar control is the simplest form of using body tracking data for interaction. The tracked joint positions and/or orientations are simply mapped onto a virtual avatar and therefore, all user actions are directly mimicked by the avatar. This technique is present in most commercial full

body interaction games, at least as a feedback mechanism. It is also a part of FUBI's Horde3D GameEngine integration. To make the mapping between user joints and avatar joints more visually appealing, the avatar control has to take the different body measurements of the avatar and the actual user into account. This can either be achieved with an initial calibration step, or inverse kinematics can be used to correct the user movements for the virtual skeleton on the fly. With a working avatar control, users can move the avatar within its virtual environment and change its joint configuration. The system can further easily detect when the user reaches specific places in the virtual environment by checking the avatar position against the point of interest. In a similar way, the positions of the hands or other body parts can be tested against specific game objects to activate interaction with them.

However, this kind of interaction has quite some restrictions. At first, navigational space in the virtual environment would be limited to the space covered by the depth sensor's field of view which dramatically restricts the player movements. When using a stationary display, the user can also loose focus on the display when moving in other directions. The avatar control can enable to interact with virtual objects by touching them, however, actual interaction as in real-life is quite challenging to achieve with the incomplete virtual representation of the user (the Kinect usually only tracks single points for the hands) and without haptics.

Most of the commercially available full body interaction games include a user controlled avatar. However, to reduced the shortcomings of this technique, users usually do not navigate the avatar directly through the virtual world. In most games, the avatar moves autonomously and users can only control aspects of the movement. For example, the avatars in the cars of Kinect Joy Ride automatically accelerate and users only have to steer left and right with an invisible steering wheel. In Kinect Sports, users get, among others, the task to take part in a 100-meters race. Users can therefore influence the speed of their avatars by running without moving in front of the Kinect, but cannot turn left or right.

A slightly different approach has been proposed by Bleiweiss et al. [15] who combine the avatar control with predefined animations. They only

apply parts of the user's skeleton tracking directly on their avatars, while other parts are blended with predefined animations and some parts are even completely replaced by those animations. The predefined animations are triggered according to the current game situation and by recognizing specific user behavior. For example, a sliding animation is applied when the avatar is sliding down a chute or a jumping animation is triggered when a user jump is recognized. In this way, specific animations better fit to the avatar and they can exaggerate the player's motions up to permitting supernatural movements of the avatars. Further, spatial restrictions can be resolved by e.g. exchanging a walking in place gestures by actual moving forward. However, it requires both, recognition technology to detect the user behavior and blending techniques to smoothly switch between the skeleton tracking and predefined animations.

We propose a different approach of how the users can navigate their avatar through the virtual scene. Instead of requiring poses, such as leaning forward or walking without moving, we are trying to achieve a more natural way of interaction by requiring the users to move in the real world themselves. However, the virtual navigation space for the avatar would then be limited to the real space in front of the screen. For this reason, we scale up the users' translation for the virtual characters (i.e. one step of the user results in multiple steps of the avatar) to cover a larger virtual navigation space. To enable more natural movements of the virtual agent, the users' body orientation is applied to the character: If users turn left or right, their avatars will do the same. In this vein, users can orient their avatar towards other agents during a conversation. In addition, the characters have predefined animations for walking forward, backward, left or right, that are applied according to the orientation and movement. This approach is similar to the one presented by Bleiweiss et al. [15]. However, they focus on exaggerating the users' motions and do not map the movement of the whole human body onto the characters. As the legs of the virtual character are animated separately, we further only need to apply the tracked joint positions of the user's upper body (mainly the arms) to the bones of the virtual agent. We hope that this increases the users' sense of immersion and

their identification with the character. An implementation and evaluation of our approach will be described in Section 7.1.

5.3 Freehand GUI Interaction

The next type of full body interaction implemented in the FUBI framework is freehand GUI interaction which will be investigated closer in the following. There are several application scenarios for freehand GUI interaction, and – depending on the scenario – the realization can be more or less similar to traditional mouse interaction with GUIs. One application scenario for freehand GUI interaction with a high number of on-screen items is text input using a virtual keyboard. Within this section, I will concentrate on this example for demonstrating various techniques for freehand GUI interaction and giving insides on the most important issues.

Overall, our work is most similar to the work by Ren et al. [148] which I described in Section 2.7.1. We as well use a dwell based selection and a selection via arm movements in different directions. However, we do not limit the gestural selection to the same hand that is controlling the cursor movement, but we allow for performing the selection gesture with the second hand. In the same way, we use a standard layout keyboard and a circular keyboard, but we further include a keyboard with a bimanual layout that is split to the left and right sides of the screen.

5.3.1 Virtual Keyboard Text Input

Virtual keyboard based text input requires at least three important parts. Apart from the cursor control and item selection as described in Section 2.4, virtual keyboard based text input heavily depends on the arrangement of the keyboard's characters as well. Therefore I discuss different keyboard layouts in the literature and our own layouts in the next section. In Sections 5.3.1 and 5.3.1, I describe our implementations regarding the cursor control and item selection.

Keyboard Layouts

Different arrangements for dedicated devices have been investigated including the traditional QWERTY layout as well as alphabetical layouts, or new arrangements, like the OPTI layout [114], 3D layouts as presented by Shoemaker et al. [161], or layouts with a selection area (cf. Quikwriting [143] Cirrin [116] or GesText [81]). Several methods for improving the design of virtual keyboards have been presented in the literature. One idea is to arrange letters in such a way that the average travel distance between them is minimized when writing words of a language. While Dunlop and Levine [42] presented a method to determine which character should preferably be placed on which key, Zhai et al. [197] optimized the position of the keys themselves. Findlater and Wobbrock [48] additionally improved the size of the keys. Furthermore, a number of researchers proposed to separate the key selection in two parts in order to enable the users to input more characters than keys are available, see, for example, the numeric keypad based method presented by Ingmarsson et al. [76] for multimedia home terminals or the bimanual interaction techniques proposed by Don and Smith [40] for virtual keyboards on multi touch displays. Overall, the previous work suggests that it is worthwhile to adapt keyboard layouts for scenarios in which the interaction devices and modalities are restricted in some way. Usually, new keyboard layouts are not able to achieve better results than a standard layout at the beginning, but they can outperform it after a number of training sessions because of their optimized design (cf. McKenzie and Zhang [114], Zhai et al. [197] or Dunlop and Levine [42]).

To investigate how different keyboard layouts can be applied for freehand interaction, we investigated the following three layouts:

The first one uses a QWERTZ layout (top of Figure 5.10). However, our keyboard only consists of the alphabetic lower case letters “a” to “z” and additional a space bar, a return key (symbolized with a down arrow), and a backspace key (symbolized with a left arrow).

We also developed a new keyboard layout especially designed for the mid-air interaction that is displayed in the left bottom of Figure 5.10. As the usage of both hands has proven promising in our initial evaluation study

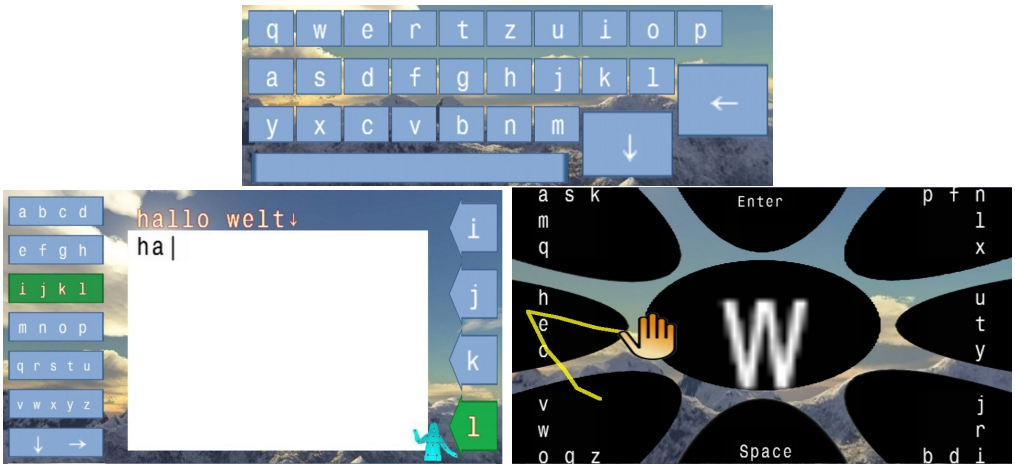


Figure 5.10: Keyboard layouts used in our freehand text input systems

this method is designed for two hand interaction and separates the keyboard to the left-hand and right-hand side of the screen. The left keyboard part includes the same keys as our QWERTZ keyboard, but it groups them into seven groups. The right keyboard part always displays the single letters of one group of the left keyboard part. This separation is similar to the one used by Don and Smith [40]. We decided to use an alphabetical arrangement of the characters, both for the groups and for the single characters for not further complicating the new layout, and in the hope to provide an easier start for novice users.

The last used keyboard layout is a modified version of the Quikwriting keyboard by Perlin [143] that arranges the letters circularly around a center area (see right bottom of Figure 5.10). The letters form groups in eight equally sized sectors around the center, and they are also sorted in a way that frequent characters can be written faster as suggested by Perlin.

Cursor Control

Various options on how to map the hand to cursor movement have been described in Section 2.4.1. For our text input methods, we decided against a ray-casting technique as its accuracy and comfort of hand position would be dependent on the placement and dimensions of the screen (cf. Vogel an Balakrishnan [182]). Instead we apply an indirect mapping by taking

the current x and y coordinates of the vector from the shoulder to the hand joint and mapping it to screen coordinates (x_s, y_s). The origin of screen coordinates (upper left screen corner) is mapped to a specific value (x_0, y_0) and a *width* and *height* in tracking coordinates is defined to cover the complete screen. This results in the mapping formulas $x_s = \frac{x-x_0}{width}$ and $y_s = -\frac{y-y_0}{height}$. Note that the y is flipped for screen coordinates. For one of our text input methods we further use spherical coordinates (r, ϕ, and θ) that have their origin at the shoulder position. We then take the angles ϕ and θ of the vector to the hand to determine screen coordinates. ϕ is mapped to x_s and θ to y_s in the same manner as described before. The spherical coordinates have the advantage that ϕ and θ do not change if the user moves the hand in direction of the shoulder-hand vector. As the items of a virtual keyboard are usually quite close to each other, we do not consider an area cursor as proposed by Grossman and Balakrishnan [59] or Su et al. [168] that would only add ambiguity.

As the raw tracking coordinates are rather noisy (cf. Koshelham and Sander [89] or Kopper et al. [100]), we need a filter for smoothing the cursor movement. A simple low-pass filter removes jitter, but also implies a significant delay for faster movements. To avoid these problems, we use a modified version of the adaptive low-pass filter by Casiez et al. [24] that smooths the cursor's movement depending on its current speed. This results in a still cursor when the hand is not moving despite of the jitter in the tracking coordinates. In addition, it allows fast and large movements of the cursor without noticeable delays.

As the user does not get any haptic feedback while typing with the Kinect "in the air", it is important to give feedback on other channels. Therefore, we display a cursor on screen in most of our virtual keyboard based systems. Figure 5.11 on the left-hand side displays the graphical representations of the default cursor. In addition, we use visual and auditive feedback for both hovering over a key and selecting it, i.e., the key changes its color accompanied by a sound on

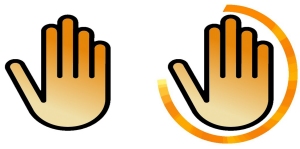


Figure 5.11: Cursors for our freehand text input

hover, and does the same with a different color and sound after selection.

Key Selection

The next task is to distinguish between pointing with and without an actual selection. As already pointed out in Section 2.4.2, this task is not trivial e.g. in comparison to methods in which the users hold a device in their hands and can simply press a button for the selection. To this end, we use three main methods to achieve the key selection: selection by dwell, selection by moving to a selection area, and selection by a short pushing gesture.

Selection by dwell in combination with our QWERTZ keyboard layout forms our first text input method (= *dwell*). The user has to hover the cursor over a key of the virtual keyboard until it is automatically selected. A loading ring around the cursor (see Figure 5.11 on the right-hand side) represents the progress of the dwell time. A dwell time of 1.2 seconds has been proven useful to enable comfortable writing for non-expert users as in our first evaluation study. It is the same value used by Ren et al. [148] which is slightly below the one in the Microsoft Xbox Kinect GUI of about 1.3 seconds. As the loading circle around the cursor on the right-hand side of Figure 5.11 is three quarters filled, it is displayed in this manner after hovering the cursor over a key for 0.9 seconds.

Selection by moving to a selection area is used with our Quikwriting keyboard (= *quikwriting*). We use the same cursor control mechanism as described before, but we add a yellow line to display a trace of the cursor. The key selection is similar to Jones et al. [81], but the actual process is adopted more directly from Perlin [143]. To write a character, the user has to move the cursor from the center area to the sector that contains the wanted character which first selects the middle character of the group this sector is containing. Thereafter, the cursor has to be moved around the circle into a different sector to switch between the single characters of the initially selected group. For finally writing the currently selected character, the user has to move the cursor back to the center. For example the lower left sector includes the letters “v”, “w”, “o”, “g”, and “z”. If the cursor is moved from the center into this sector, this group of characters is chosen,

initially selecting the "o". If the user wants to write a "w", the cursor has to be moved one section further in clockwise direction before returning to the center, two sections for the "v". For an "g" the cursor has to be moved one section in counter-clockwise direction, two sections for a "z". For feedforward, we always display the currently selected character in the center area, so the users always know which character would be entered if returning to the center (in Figure 5.10 a "w").

Selection by a short pushing gesture is used with our QWERTZ layout, but also with our new layout that separates the keyboard on the left-hand and right-hand side of the screen. This selection method tries to reduce the time needed for dwell based selection by using a pushing gesture that can be performed faster than the dwell time. We therefore measure the velocity of the user's hand in relation to the shoulder and wait that it exceeds a certain velocity while the cursor is hovering over a key. In pretests, a pushing velocity of 1 m/s has proven useful, as it is low enough to allow a small and as least exhausting as possible movement for the push gesture. In addition it is high enough to prevent recognition of unintended movements as pushing. With the QWERTZ layout, we use three different pushing gestures that form the next three types of our virtual keyboard based text input methods:

1. Pushing shortly in direction of the screen (= −z-direction) with the cursor controlling hand (= *z-push*). This is intuitive as the pushing direction is towards the screen that displays the keyboard, however, it is a challenging task to keep the cursor hovering over a key while performing the pushing gesture. To tackle this problem, the cursor is already pinned at its current position as soon as half of the pushing velocity is reached.

2. Pushing shortly in the current pointing direction with the cursor controlling hand (= *dir-push*). For this method we are using the cursor control with spherical coordinates as described in the previous section. The idea behind this method is, that a pushing gesture in the current pointing direction should be physiologically easier to perform and some early tests indicated that it could be a more natural move-

ment. This method still has some kind of problem with the difficulty to keep the cursor at the same position while pushing. Therefore, it also pins the cursor at its current position as soon as half of the pushing velocity is reached.

3. Pushing shortly downwards (= −y-direction) with the second hand (= *other-push*). The separation of the pointing and selection on two different hands is done to avoid errors that are caused by moving the cursor away to an unwanted position during the execution of the pushing gesture with the same hand. The method has the drawback that you need to have both hands free and you need to coordinate the actions of both hands.

For our new keyboard layout, we use a pushing gesture with the right hand to the left (= −x-direction) which is usually called a swipe in surface computing, and this forms our last virtual keyboard based text input method (= *swipe*). This time, both hands are used for switching between different letters by moving them along the y-axis. In particular, users can choose between different groups of characters by moving the left hand up and down, and they can switch between the single characters of this group by moving the right hand up and down. For finally entering one character the users have to perform the aforementioned swiping gesture to the left with the right hand. We do not display cursors for this method, as having two different cursors on the screen (one for each hand) would probably be more confusing than helpful and there is no actual pointing happening, but a character is indicated by the y-coordinates of the two hands only. The principle of bimanual input is more or less adopted from Don and Smith [40]. The key selection mechanism with the swiping gesture is inspired by the GUI interaction as realized in the commercial game "Dance Central". We consider it as a promising mechanism as users only need one axis for indicating a character and a second axis for selecting it. Our hope is that it should be easier to use than the methods in which a pushing gesture has to be performed with the cursor controlling hand, and the hand therefore has to be controlled in three axes at the same time.

In Section 7.2, I will describe two different studies in which we compared

the presented techniques for freehand GUI interaction.

5.4 Full Body Gesture Recognition

The gesture recognition system is the main part of the Full Body Interaction framework (FUBI). Dependent on the used tracking software, we get positions and orientations for up to 25 different body joints and optionally the ten finger positions. The joint data is analyzed in the recognition framework for detecting gestures that are defined in XML files using the FUBI gesture XML language.

Listing 5.1: Structure for the FUBI gesture XML language

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?>
2  <FubiRecognizers ... xsi:schemaLocation="..http://hcm-␣
       lab.de/downloads/FubiRecognizers.xsd">
3    <BasicRecognizer name="X">
4      ...
5    </BasicRecognizer>
6    ...
7    <BasicRecognizer name="Y">
8      ...
9    </BasicRecognizer>
10   ...
11   <CombinationRecognizer name="Z">
12     ...
13     <Recognizer name="X"/>
14     ...
15     <Recognizer name="Y"/>
16     ...
17   </CombinationRecognizer>
18   ...
19 </FubiRecognizers>
```

Files using the FUBI gesture XML language are basically structured as depicted in Listing 5.1. The XML scheme is described in the FubiRecognizers.xsd which can be found in Section A.1. The root node is called "FubiRecognizers" (line 2). Below the root, the definitions of the actual recognizers follow, including basic recognizer (lines 4–15) as well as combination recognizers referring to the basic ones (lines 17–20).

FUBI supports five types of basic (rule-based) recognizers which can be grouped into three different static postures and two dynamic gestures

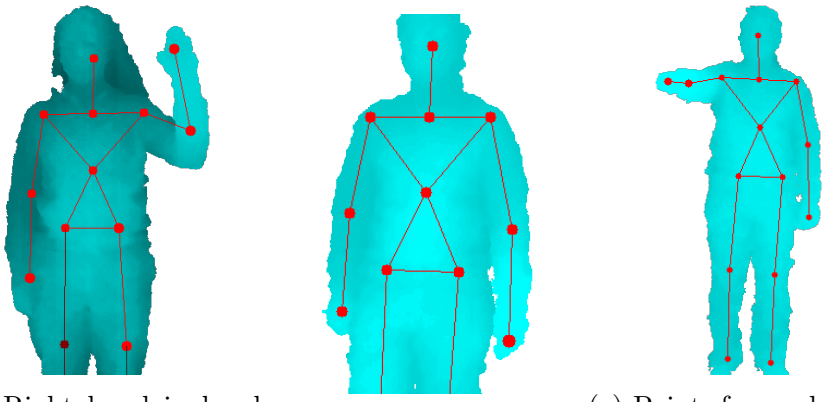
regarding the form dimension of our taxonomy (cf. 4.1.2). The additional subcategories are mainly due to whether the recognizer observes the position or the orientation the joints. Therefore, static postures are either defined by a specific position of one to three joints (= **JointRelationRecognizer**), the orientation of a joint (= **JointOrientationRecognizer**), or the number of displayed fingers of one hand (= **FingerCountRecognizer**). The latter one can be seen as a special case of joint relation, however, it is recognized separately because of technical reasons. The dynamic gestures either look at the change in position of one or two joints (= **LinearMovementRecognizer**), or they look at the change in orientation of one joint (= **AngularMovementRecognizer**). The body parts dimension of the recognizers is determined by the joint(s) that they are observing. The gesture type dimension is irrelevant for the recognition of a gesture and therefore not included in the FUBI gesture XML language, however, it is important to its meaning. Apart from the five rule-based recognizers, FUBI allows to use template-matching recognizers that compare input gestures against template gestures according to their geometrical gesture path. In the XML, they are an additional type of basic recognizer which is called **TemplateRecognizer** and refers to an additional file which includes the template as one or more files of recorded tracking data. The five rule-based basic recognizers and the template recognizers can further be combined to sequences within a **CombinationRecognizer** which basis is an FSM.

The rule-base recognizers were chosen to efficiently detect postures or simple motions. Nevertheless, they can be combined in the CombinationRecognizers to already cover a large part of possible gesture types. Last but not least, the TemplateRecognizers were added to cover the remaining cases in which it is impossible or at least impractical to use the other recognizer types. They include the most complex gesture recognition algorithms to make them as powerful as possible, however at the cost of performance. Therefore they are only meant to be used as a kind of last resort.

In FUBI's XMl definition, all of the basic recognizers can require a certain confidence value of the tracking, they can use raw or filtered data and local or global positions/orientations. The following sections describe

the different recognizers in detail including all additional configuration options and sample gestures with descriptive images and the corresponding XML code of the FUBI gesture definition language. All samples are using body joints of the default body tracking sensors (XML nodes "Joint" and "Joints"). However, one can also choose the hand joints of the finger tracking sensors by using the XML nodes "HandJoint" and "HandJoints".

5.4.1 Joint Relations



(a) Right hand in head height

(b) User is close

(c) Point forward with left hand

Figure 5.12: Joint relation sample gestures

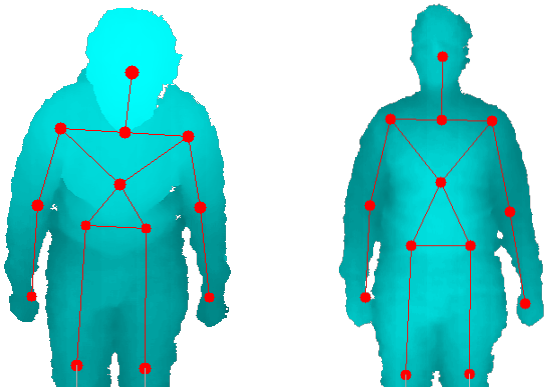
Joint relation recognizers look at the position of a (*main*) joint in relation to another (*relative*) one or relative to the sensor. For example, they test, whether a joint is above other joints or how far a joint is away from the sensor. Therefore the relations define minima and maxima for the values of each axis or the overall distance (lines 1–5). Alternatively, one can describe the relation by adding one or more of the basic relations: *inFrontOf*, *behind*, *leftOf*, *rightOf*, *above*, *below*, or *apartOf* with thresholds for the minimum and maximum (lines 6–9). To make the recognizers more robust against different users, one can – apart of the default millimeters – choose user-dependent measuring units such as body height or shoulder width for defining the relations (line 10). Another powerful option is to select an additional third joint (*MiddleJoint*), which is related to the line

between the other two joints. This is useful, e.g. for detecting a stretched arm, by requiring the elbow to be in line with the hand and shoulder joint (lines 14–17).

Listing 5.2: XML samples for joint relation recognizers

```
1  <JointRelationRecognizer name="RightHandInHeadHeight">
2    <Joints main="rightHand" relative="head"/>
3    <MaxValues y="150"/>
4    <MinValues y="-150"/>
5  </JointRelationRecognizer>
6  <JointRelationRecognizer name="UserCloseToSensor">
7    <Joints main="torso"/>
8    <Relation type="apartOf" max="1200"/>
9  </JointRelationRecognizer>
10 <JointRelationRecognizer name="PointingLeft" measuringUnit="armLength">
11   <Joints main="leftHand" relative="leftShoulder"/>
12   <Relation type="inFrontOf" min="0.25"/>
13   <Relation type="below" max="0.5"/>
14   <MiddleJoint>
15     <Joint name="leftElbow"/>
16     <Relation type="apartOf" max="0.25"/>
17   </MiddleJoint>
18 </JointRelationRecognizer>
```

5.4.2 Joint Orientations



(a) Bowing position (b) Standing straight

Figure 5.13: Joint orientation sample gestures

Joint orientation recognizers are defined by a minimum and/ or a maximum angle for the orientation of a specific joint around an axis similar to

degrees of freedom in joint animations (lines 1–5). Alternatively, they can be defined by a concrete orientation (all three axis) and a tolerance value which defines the maximum difference between the desired and the actual orientation (lines 6–9).

Listing 5.3: XML samples for joint orientation recognizers

```
1 <JointOrientationRecognizer name="Bow">
2   <Joint name="torso"/>
3   <MaxDegrees x="-20"/>
4   <MinDegrees x="-180"/>
5 </JointOrientationRecognizer>
6 <JointOrientationRecognizer name="TorsoStraight">
7   <Joint name="torso"/>
8   <Orientation x="0" y="0" z="0" maxAngleDifference="20"/>
9 </JointOrientationRecognizer>
```

5.4.3 Finger Counts

Finger count recognizers attempt to detect a minimum and/or a maximum number of displayed fingers of the left or right hand. Again, one can define a minimum and maximum for that number (line 3). To get a more stable recognition of numbers, one can activate median filtering (line 3). By setting a larger window size for the filtering, outliers are ignored more effectively, however, this results in a noticeable delay as well.

To enable the finger count recognition, FUBI enhances the tracking provided by the different sensors by performing computer vision operations on the depth image. At first, FUBI crops the depth image around the tracked position of the hand to be investigated and also applies a threshold on the depth values using the depth of the hand. In this way, mainly the hand shape remains in the image. Extracting the largest contour in the image further removes artifacts coming from small pieces of other body parts of similar depth and artifacts caused by noise in the depth image or tracking errors. After

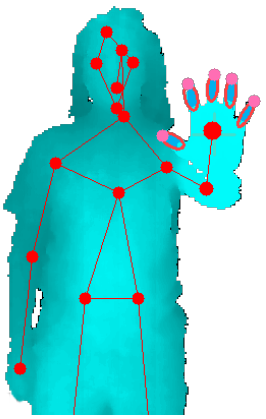


Figure 5.14: Finger count sample gesture: open right hand

that, only the hand and possibly a small part of the arm remain in the cropped image (cf. Figure 5.15a).

Listing 5.4: XML sample for a finger count recognizer

```
<FingerCountRecognizer name="OpenRightHand">
  <Joint name="rightHand"/>
  <FingerCount min="3" useMedianCalculation="1" medianWindowSize="8"/>
</FingerCountRecognizer>
```

Before applying the actual finger detection on the image, a median blur is applied for fixing smaller holes in the extracted hand contour. The process of extracting the shapes of extended fingers is based on the work by Kang et al. [84].

(a) Hand shape extracted (b) Morphed image calculated (c) Morphed image subtracted (d) Detected finger shapes

Figure 5.15: Process for finger tracking on the depth image

At first, a morphological opening operation is applied on the image in order to extract the palm (cf. Figure 5.15b). Subtracting the extracted palm from the original image, leaves only the finger shapes and some additional small artifacts (cf. Figure 5.15c). The artifacts are removed by applying another contour detection and requiring a minimum size. In this way only the contours of the finger remain and can be counted. To make the process visible to the user, FUBI provides an option for rendering the detected finger shapes on the depth image as presented in Section 5.1. For this purpose, FUBI additionally computes the center of palm, fits ellipses around the detected finger shapes and labels the ends of the ellipses that are further away from the center as finger tips (cf. Figure 5.15d).

In a quick accuracy test with 35 hand images, we achieved a better recognition accuracy using this approach with 94% correct detections in comparison to 83% correct detections using the convexity defects method adopted from Bailly et al. [8].

While the current technique already offers a first way to integrate finger interaction with a low-quality depth image as provided by the Kinect, more advanced techniques [159] could significantly increase the interaction possibilities in the future.

5.4.4 Linear Movements

Linear movement recognizers are defined by the change of position of a joint in a specific direction and a minimum and/or a maximum speed. The direction can be specified with an actual movement vector defining values for each axis (line 3), or by choosing a basic movement direction, i.e. *left*, *right*, *up*, *down*, *forward*, *backward*, or *anyDirection* (line 8). Setting *anyDirection* or omitting the direction completely results in analysis of the speed for movement in any direction. In addition, a maximum for the angular difference between the desired and the actual movement direction can be defined (line 3). By default, the recognizer compares the full speed of the movement against the speed restrictions, however, one can also define that it should only take the speed of the vector component into account that is pointing in the desired direction (line 1). In this way, the recognizer gets much more sensitive to slight differences in direction. Instead or in addition to the speed restriction, one can also define a length restriction for the movement with minimum and maximum defined in millimeters or other measuring units as with the joint relations. However, note, that this restriction only works when using the recognizer within a combination. Only here, the basic recognizer gets more information about

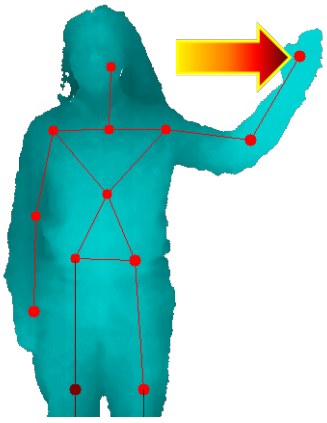


Figure 5.16: Linear movement sample gesture: right hand moves right

joint positions previous to the last frame. To make a recognizer invisible outside of combinations, it can be marked as *hidden*. This is also working for all other basic recognizers.

Listing 5.5: XML samples for linear movement recognizers

```
<LinearMovementRecognizer name="RightHandMovesRight"
    useOnlyCorrectDirectionComponent="true">
  <Joints main="rightHand" relative="rightShoulder"/>
  <Direction x="1" y="0" z="0" maxAngleDifference="30"/>
  <Speed min="1200"/>
</LinearMovementRecognizer>
<LinearMovementRecognizer name="LeftHandStill">
  <Joints main="leftHand" relative="leftShoulder"/>
  <BasicDirection type="anyDirection"/>
  <Speed max="300"/>
</LinearMovementRecognizer>
<LinearMovementRecognizer name="leftHandDown" visibility="hidden">
  <Joints main="leftHand"/>
  <BasicDirection type="down"/>
  <Length min="1" measuringUnit="upperArmLength"/>
</LinearMovementRecognizer>
```

5.4.5 Angular Movements

Listing 5.6: XML sample for an angular movement recognizer

```
<AngularMovementRecognizer name="HeadDown" useFilteredData="true">
  <Joint name="head"/>
  <BasicAngularVelocity type="pitchDown" min="15"/>
</AngularMovementRecognizer>
```

Angular movement recognizers are defined by the change of the joints orientation, which can be requested to stay within a certain minimum and/or maximum around each of the three coordinate axes. Similar to the linear movements, the angular directions can also be configured by choosing minima and/or maxima for basic turning directions, which here describe the rotational motion around one axis

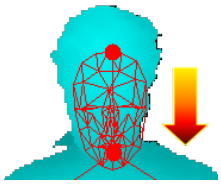
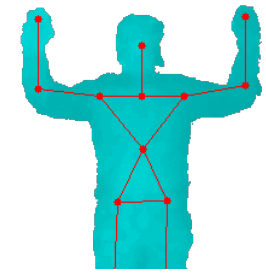


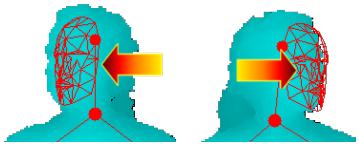
Figure 5.17: Angular movement sample gesture: head pitches down

in a certain direction according to the user's view, i.e. *rollLeft*, *rollRight*, *pitchUp*, *pitchDown*, *yawLeft*, or *yawRight* (line 3).

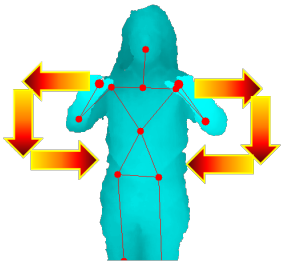
5.4.6 Combinations



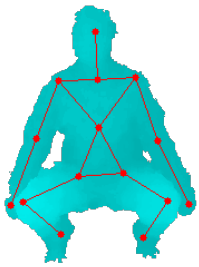
(a) Both hands hold up



(b) Head shake



(c) Forming a chest with the hands



(d) Squatting

Figure 5.18: Combination sample gestures

A combination recognizer defines a finite state machine that consists of one or more states which contain sets of the above mentioned basic recognizers. Each state can define a minimum and/or a maximum duration, which the recognizers have to fulfill in the recognition process to get into and stay in the that state (sample b and c). As temporary tracking errors can sometimes cause the cancellation of a state, a maximum interruption time can be defined which allows the state to be temporarily interrupted (sample c). Furthermore, the transition to the next state can as well be restricted with a maximum translation time (sample b and c).

Figure 5.19 displays the automaton for the "forming a chest" gesture of sample c. While the three blue states represent the main states with sets of basic recognizers as defined in the XML, the two green states are implicitly given start and end states, and the five red states model the transition and interruption restrictions. The transition edges contain three types of

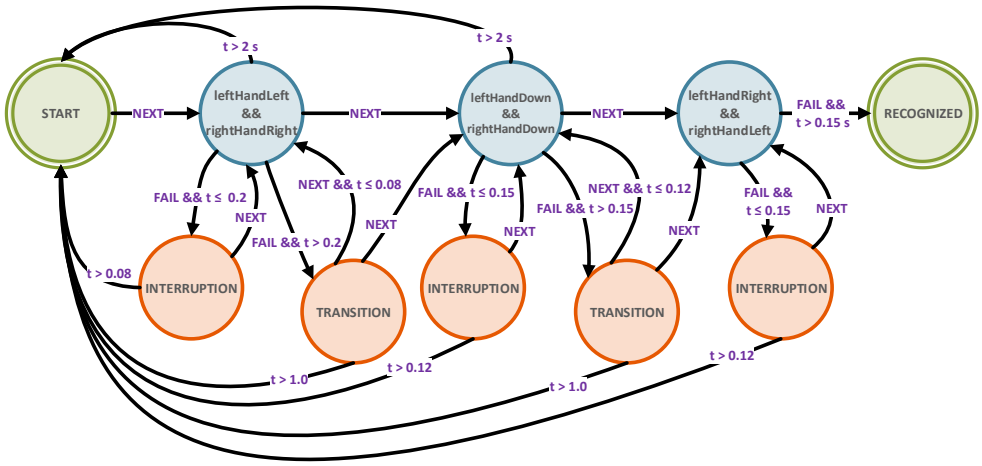


Figure 5.19: Combination recognizer of Figure 5.18c modeled as a finite state machine

conditions concatenated with logical operators: "FAIL" means that the recognizers of the current state failed; "NEXT" means that the recognizers of the next state are fulfilled, and "$t \bowtie x$; $with \bowtie \in \{<, \geq\}$" requires that the duration t within the current state has or has not passed x. Already in this small example, it can be seen that completely modeling the recognizer as an automaton can get more complex than defining it in the FUBI XML gesture definition language. The recognizers within one state are by default concatenated with a logical "and" (sample a, lines 4–5), which means all of them have to be fulfilled during the duration of that state. However, they can also be negated, which means they are not allowed to be fulfilled (sample a, lines 19 and 22), and they can also be complemented by another set of recognizers concatenated with a logical "or", which means it is enough when one of the groups is fulfilled (sample a, lines 11–13 and 20–23). The combination recognizer has several additional optional attributes that evolved from specific requirements in certain situations. By default, the combination recognizer succeed in a recognition as soon as the minimum duration of the last state is fulfilled. However, in some cases, it is important that the combination is only reported to be finished, when the gestures of last state are actually stopped by the user (sample c, line 2).

Similarly, FUBI allows short interruptions during the whole duration of a state. However, it is sometimes important that there is no interruption while waiting for the minimum duration (sample c, line 3). Especially when using the data of less reliable joints, it can be useful to enable the *ignoreOnTrackingError* attribute for one or more recognizers (sample d). In this case, the recognizer will be ignored that a recognizer is not fulfilled when it reports a tracking error. Nevertheless, at least one recognizer per state needs to be fulfilled all the time. Another way to make the recognizer less restrictive is to change the on fail behavior so that the combination will not be aborted completely, but only go back one state in case a state fails (sample b). Especially for repetitive movements as in a head shake or hand wave, this allows much more variations of the gesture. However, it can introduce more false recognitions as well.

Listing 5.7: XML samples for combination recognizers: (a) different conjunctions for the hands in head height

```
1  <!--...joint relations for RightHandInHeadHeight and LeftHandInHeadHeight need
       to be defined first, then...-->
2  <CombinationRecognizer name="BothHandsInHeadHeight">
3    <State>
4      <Recognizer name="RightHandInHeadHeight"/>
5      <Recognizer name="LeftHandInHeadHeight"/>
6    </State>
7  </CombinationRecognizer>
8  <CombinationRecognizer name="AtLeastOneHandInHeadHeight">
9    <State>
10     <Recognizer name="RightHandInHeadHeight"/>
11     <AlternativeRecognizers>
12       <Recognizer name="LeftHandInHeadHeight"/>
13     </AlternativeRecognizers>
14   </State>
15 </CombinationRecognizer>
16 <CombinationRecognizer name="OnlyOneHandInHeadHeight">
17   <State>
18     <Recognizer name="RightHandInHeadHeight"/>
19     <NotRecognizer name="LeftHandInHeadHeight"/>
20     <AlternativeRecognizers>
21       <Recognizer name="LeftHandInHeadHeight"/>
22       <NotRecognizer name="RightHandInHeadHeight"/>
23     </AlternativeRecognizers>
24   </State>
25 </CombinationRecognizer>
```

Listing 5.8: XML samples for combination recognizers: (b) a head shake

```
<!--...angular movements for HeadLeft and HeadRight need to be defined first,
    then...-->
<CombinationRecognizer name="HeadShake">
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadLeft"/>
  </State>
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5"
      onFail="goBack">
    <Recognizer name="HeadRight"/>
  </State>
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5"
      onFail="goBack">
    <Recognizer name="HeadLeft"/>
  </State>
  <State minDuration="0.05" onFail="goBack">
    <Recognizer name="HeadRight"/>
  </State>
</CombinationRecognizer>
```

Listing 5.9: XML samples for combination recognizers: (c) forming a chest

```
<!--...linear movements for leftHandLeft, rightHandRight, ... need to be defined
    first, then...-->
<CombinationRecognizer name="Chest" waitUntilLastStateRecognizersStop="true">
  <State maxDuration="2" minDuration="0.2" maxInterruptionTime="0.08"
      timeForTransition="1" noInterruptionBeforeMinDuration="true">
    <Recognizer name="rightHandRight"/>
    <Recognizer name="leftHandLeft"/>
  </State>
  <State maxDuration="2" minDuration="0.15" maxInterruptionTime="0.12"
      timeForTransition="1">
    <Recognizer name="rightHandDown"/>
    <Recognizer name="leftHandDown"/>
  </State>
  <State minDuration="0.15" maxInterruptionTime="0.12">
    <Recognizer name="rightHandLeft"/>
    <Recognizer name="leftHandRight"/>
  </State>
</CombinationRecognizer>
```

Listing 5.10: XML samples for combination recognizers: (d) squatting

```
<!--...joint orientations for RightKneeBent, LeftKneeBent need to be defined
    first, then...-->
<CombinationRecognizer name="Squat">
  <State minDuration="0.5" maxInterruptionTime="0.15">
```

```
4     <Recognizer name="RightKneeBent" minConfidence="0.75" useFilteredData="true"
          ignoreOnTrackingError="true"/>
5     <Recognizer name="LeftKneeBent" minConfidence="0.75" useFilteredData="true"
          ignoreOnTrackingError="true"/>
6   </State>
7 </CombinationRecognizer>
```

5.4.7 Template-Based Symbolic Gestures

In the case that the gesture consists of a more complex shape, i.e. a symbolic gesture such as the circle in Figure 5.20, it can be impractical to define the gesture using the composite approach as described in the previous section. Therefore, FUBI also supports to apply a template-matching approach for recognizing those gestures. As the tracking data does not provide temporal segmentation, the application of a template-matching approach requires a brute force algorithm in which the recognition is applied in each frame on a certain window of buffered data. This can lead to massive performance problems when a large number of gestures should be classified in real-time. Furthermore, recognition algorithms that yield accurate results with mouse or touch input, can perform much less robust with full body tracking data because of the unsegmented and more noisy data. Therefore, I recommend to only use this technique in the case the other methods cannot be applied easily.

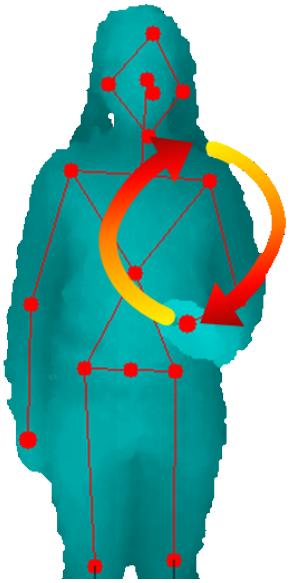


Figure 5.20: Template recognizer sample gesture: drawing a circle

The template recognizers apply a geometrical gesture recognition similar to the $1 recognizer [190]. In addition, dynamic time warping [128] can be applied, and the templates can be trained with multiple samples by calculating raw means and covariance matrices, or using Gaussian mixture models and regression [23].

In the FUBI Recognizer XML, template-based symbolic gestures are integrated as in Listing 5.11. Technically speaking, they are basic recognizers and therefore, they can define several at-

tributes common to the other basic recognizers such as a certain confidence value of the tracking, the choice between raw or filtered data and local or global positions/orientations.

The most important part of the recognizer are the references to the training data (e.g. line 3), which refer to separate files that include recorded tracking data. As the gesture is usually only represented by a part of that data, one can set the start and end of the gestures within the files using the frame ids. Sometimes, it is helpful to use only 2D or even 1D data depending on the dimensionality of the gesture. Therefore the *IgnoreAxes* node allows to reduce the used dimensions of the data (line 7). In addition, one can choose the joints that are used by one or multiple Joints nodes as known from the joint relation recognizers (line 2).

The TemplateRecognizer node itself, can have the following additional attributes: *useOrientations* is set to use orientations instead of position. *distanceMeasure* defines how the recognizer calculates the distance between the input gesture and the template either using an *euclidean*, *manhattan*, *turningAngle* difference, or *malhanobis* distance, while the *maxDistance* option defines a maximum for this distance. Regarding the normalization steps adopted from the $1 recognizer, one can set *maxRotation* to reduce the rotation invariance and *aspectInvariant* to reduce the scale invariance, while the resampling can be changed with *resamplingTechnique*. *maxRotation* defines the maximum a gesture candidate is allowed to be rotated to match the indicative orientation of the template gesture. Therefore a smaller value requires to perform the gesture with a rotation more similar to the template, while a very large value allows completely different orientations of the gesture, e.g. an arrow pointing to the left or right. When the *aspectInvariant* attribute is set to true, the scale normalization is applied in the same way as in the one dollar recognizer and the gesture is scaled to a unit cube. If the attribute is set to false, the gesture only gets scaled uniformly along the axis with only the longest side being scaled to length one. In the latter case, e.g. a rectangle can be distinguished from a square or an ellipse from a circle. Dynamic time warping is not seen as an own recognition technique, but it can additionally be activated with the *useDTW* attribute. As an

additional enhancement, the attribute *maxWarpingFactor* further defines how much warping should be allowed measured in relation to the whole gesture length. The previously mentions Gaussian mixture regression can be applied by defining the attribute *stochasticModel* and setting the number of states with *numGMRStates*. As the TemplateRecognizers apply the matching on a window with fixed length, they often do not segment the data perfectly. Therefore the attribute *searchBestInputLength* additionally activates that the recognizer investigates multiple window lengths using a golden section search strategy to find an approximately optimal length with few iterations. For resampling, FUBI offers a Hermite spline resampling or polyline resampling [54] in addition to the $1 equidistant one, and the resampling can be disabled completely as well.

As the TemplateRecognizers are a form of basic recognizer, they can as well be referenced within combinations. This helps, for additionally requiring a certain posture while performing the gesture, additional surrounding or accompanying gestures (even additional TemplateRecognizers).

Listing 5.11: XML samples for template recognizers

```
1  <TemplateRecognizer name="arrow" aspectInvariant="true" maxRotation="60"
       resamplingTechnique="HermiteSpline" numGMRStates="4" maxDistance="2"
       distanceMeasure="malhanobis" stochasticModel="GMR" useDTW="true">
2    <Joints main="rightWrist" relative="torso" />
3    <TrainingData file="trainingData/Arrow1.xml" start="0" end="68"/>
4    <TrainingData file="trainingData/Arrow2.xml"/>
5    <TrainingData file="trainingData/Arrow3.xml"/>
6    <TrainingData file="trainingData/Arrow4.xml"/>
7    <IgnoreAxes z="true"/>
8  </TemplateRecognizer>
9  <TemplateRecognizer name="triangle" maxDistance="0.22"
       distanceMeasure="euclidean" aspectInvariant="true" useDTW="true"
       useLocalTransformations="false">
10   <Joints main="rightHand" />
11   <TrainingData file="trainingData/Triangle.xml"/>
12   <IgnoreAxes z="true"/>
13 </TemplateRecognizer>
14 <TemplateRecognizer name="x" maxDistance="0.26">
15   <Joints main="rightHand" />
16   <TrainingData file="trainingData/X.xml"/>
17 </TemplateRecognizer>
```

5.4.8 Discussion and Conclusion

As presented in the preceding sections, FUBI incorporates a large range of different gesture recognition techniques with many configuration options. To make this usable for practitioners, the recognizers are defined in an XML language which format is accurately defined in an XML scheme. A longer example file with different recognizer definitions can be found in Section A.2. One can even avoid the need for manually editing the XML, by using the XML generator tool of the FUBI GUI described in Section 5.1.

In opposite to many classic gesture recognition approaches, FUBI allows to quickly implement full body gestures without the need for extensive training on sample data. The advantage over other similar frameworks such as FAAST [170], Kinetic Space [191] or XKin [139] is – apart from being open source – that it supports a larger amount of gesture types and provides more help for developing and integrating full body interaction in a system. While it does not take a lot of time to implement a prototype that applies full body interaction, FUBI is still powerful enough to let the developers fine-tune the interaction for the final system in a convenient way.

From another perspective, FUBI allows to implement simple gestures with simple recognizers that can efficiently be applied in a real-time interactive system. Nevertheless, it also provides a range of complexer recognition techniques to cover more complex gestures at the cost of performance. In this way, the recognition can be individualized per gesture to find the best compromise between recognition accuracy and performance.

I will present and evaluate actual implementations of gesture sets with FUBI in Chapter 7.

Chapter 6

Supporting Full Body Interaction Users

In Chapter 4, I investigated a way to design intuitive input gestures, and in Chapter 5 I described how we implemented the recognition of the gestures technically in the FUBI framework. Nevertheless, a well-designed and robustly implemented system can still be difficult to interact with for the actual user as described in Section 2.6. Therefore, it is important to provide mechanisms for supporting the users during the interaction.

The next section describes how the FUBI framework supports the full body interaction designer to provide such mechanisms and how some of the applications already described in previous sections implemented them. In Section 6.2, I describe a study that further compares different gesture visualization techniques generated out of recording of a real person.

6.1 Supporting Techniques in FUBI

FUBI incorporates multiple techniques to support the user during full body interaction, which are depicted in the following, according to four major categories. The four categories are formulated as questions a user might ask during the interaction:

1. Can I interact?
2. How can I interact?
3. Am I interacting correctly?
4. Was my interaction successful?

This approach is similar to Rich et al., who formulated six questions for their intelligent user interface Collagen [149]. However, the questions are customized for the case of full body interaction. The answers to the four questions consist of techniques for affordances and feed/-feedforward [181] and their main goal is to increase the usability of the interaction in terms of making it easier to discover, to learn, and to use.

6.1.1 Can I Interact?

The first question is actually very specific for full body interaction. As there is no explicit start of the interaction, such as picking up a controller or touching the interactive surface, users should be informed that they can actually interact with the application, i.e. the application "sees" them.

One of the easiest but efficient ways to inform the user about the current state of the tracking, is to display the depth image as an overlay on the application window. As described in Section 5.1, FUBI offers multiple possibilities on how to **display the depth image** and on how to additionally enhance it with more **detailed tracking information**. In most of the applications described in this dissertation, the whole depth image or at least the users' tracking shapes and skeletons are displayed during the complete runtime of the application to constantly tell the users about the current tracking state (cf. Figure 7.10 in the right bottom).

Of course, this can be inappropriate in some applications, as the depth image takes a considerable amount of display space and it can even distract from the actual application. In those cases, the depth image can, for example only be displayed at the beginning of the application and in case of errors, but it can be omitted during the actual interaction (cf. right-hand image of Figure 7.11). The depth image can as well be omitted in the case

of an avatar control as described in Section 5.2 because the avatar directly displays the tracking state. For not occluding the rest of the virtual world, commercial full body interaction games, such as "Kinect Adventures", often make the avatar partly transparent. In the termology by Vermeulen et al. [181], the display of depth and tracking information would be feedback about the tracking, but it as well can turn into an affordance, as it suggest to the users that they can interact by motions of their body.

6.1.2 How Can I interact?

The second goal is to tell the users how they can interact, i.e. what interaction options are available at a specific point in time (affordances/feedforward), how the users can perform the interactions (affordances), and what effect they would trigger (feedforward) in the application. Again, for the avatar control, there is usually no additional mechanism needed as the users do not need to perform any special actions, but even novice users soon discover that the avatar mirrors their movements. This is is similar for other continuous interactions as controlling a cursor with the hand (cf. the freehand cursor control of Section 5.3.1). It is usually sufficient to display the movable objects and optionally show a hint that the users should hold their hand in front to start controlling, but the interaction itself should be straight-forward.

As soon as there is more abstraction in the interaction, more information on how to interact is required. One example is the more abstract cursor control of the "swipe keyboard" described in Section 5.3.1. In the "swipe keyboard", the vertical movements of one hand are used to switch between different groups of onscreen items, while the vertical movements of the other hand switch between the elements of that group. A horizontal movement of the second hand then actually selects the currently targeted GUI item. As the interaction is divided on both hands, it is quite different to classical GUI interaction, in which only a single cursor is controlled with one hand. Therefore, the users need an introduction before they can start interacting. Nevertheless, in the mentioned case, the information on how to interact was provided by the experimenter in the introduction of the study. For

an actual product, it would be needed to embed this information in the system itself. Freehand GUI interaction includes other parts that require more helping mechanisms, e.g. highlighting and/or playing a sound when the cursor moves over an object that one can interact with. An example for affordances in freehand GUIs would be information on how to perform the actual selection gesture. Nevertheless, in the case of freehand GUI interaction, this gesture should be very short and easy to remember, so that it should again be sufficient to provide this information only at the beginning of the interaction.

With more complex body gestures, affordances and feedforward get more important, and this is also the part in which FUBI offers more mechanisms to support the interaction designer in providing this information. Those mechanisms are mainly included in the two game engine integrations of the FUBI framework. The major way to provide affordances and feedforward in FUBI is to link the input gesture with a (labeled) gesture symbol (static image or animation) as illustrated in Sections 3.1.2, and 7.4. In Section 3.2.1 an animated gesture symbol was even automatically generated out of the gesture's XML definition. At any point in time, the possible input gestures are displayed on the screen using the linked symbols (affordances), optionally enhanced with text labels that describe the corresponding system actions (feedforward). The action labels are usually not displayed if they are already represented precisely by the corresponding gestures themselves (combined affordances and feedforward), or the application is designed to hide that concrete information (hidden feedforward). Of course, those mechanisms can also be omitted in the case of frequently required interactions which are learned after a while, as the gestures used for navigation in Section 3.2. If there are too many interaction options to display them all on the screen at the same time, the symbols can be organized in a hierarchical structure as well, e.g. similar to the swipe menu in Section 7.4.2 that needs to be triggered with a specific "talk" gesture.

The automatic generation of animated gesture symbols based on virtual characters as introduced in Section 3.2.1 was further enhanced at a later stage. Generating the gesture symbols out of the Fubi gesture XML has

(a) Virtual Character
left: textured;
right: depth colorized

(b) Active Limb Highlighting

(c) Corrective Arrows

Figure 6.1: Gesture Visualizations with a Virtual Character

the advantage that the symbols do not have to be created manually, they can still be easily adapted and it is also possible to add further enhancements, e.g. for additional affordances and feedback/-forward information or a specific shading technique. The generated gesture symbol already used a shading technique that encodes depth information with colors, which is now improved to apply colorization from magenta = near to red = far (see Figure 6.1a in the right-hand image). This reasoning behind this is that one major problem for the comprehensibility of the symbols can be the missing depth perception.

As the automatic generation in Section 3.2.1 only supported very few gesture types, it was first enhanced to support all kinds of gestures that can be defined in FUBI gesture XML. In the meantime, there also multiple additional options to include information into the gesture symbol, e.g. we highlight limbs which are involved in the current gesture (see Figure 6.1b) or add arrows depicting how to move the corresponding joint for correctly performing the gesture (see Figure 6.1c). To make the generation easier to configure, optional parameters can be configured depending on the gesture type, e.g. one can define a pre-stroke posture, additional scaling, concrete timings for basic gestures, or a general tolerance value.

6.1.3 Am I Interacting Correctly?

The helping mechanisms should not stop as soon as the user starts to interact. Especially for longer gestures it may be necessary to remind the user of how the current gesture should be continued, to tell the user what gesture the system thinks he or she is performing, and in case there are still multiple candidates, to tell the user what gestures might still fit to the current input. In this way, the users can adapt their input during the performance as in the affordances and feedback/-forward mechanisms in surface computing presented in Section 2.6.

FUBI has several methods for providing affordances, feedback and feedforward during the interaction. Again, in the case of continuous interaction such as an avatar control or a freehand cursor control, there is usually no need for additional techniques as feedback is implicitly given by rendering the motion of the controlled object. Exceptions are cases, in which the application should try to prevent false postures during the actual interaction, e.g. during a rehabilitation exercise. If parts of the user's motion are interpreted as a an input event (i.e. discrete full body interaction), more information during the interaction is needed.

For assisting the developer with this task, FUBI can return so-called **correction hints** along with the boolean recognition result. Correction hints contain additional information on the current state of the recognition as well as hints on how to adapt to conform to the recognizer's requirements. Therefore, the hints provide both feedback and affordances. A correction hint always determines the main joint involved in the corresponding gesture, but the rest of the information differs depending on the type of recognizer it refers to.

Accordingly, the hint suggests to change the speed, pose, direction, form or number of displayed fingers depending on the inspected basic recognizer. Number of fingers logically refers to a **finger count recognizer** and reports how many more or less fingers the user should display. Form corresponds to symbolic gestures, i.e. FUBI's **template recognizers**. In this case, FUBI provides the current distance measured between the user performance and the gesture template. Speed, pose and direction can either refer to (the

change of) the orientation or the position of a joint which is indicated by a boolean value. They further report how much the corresponding value needs to be changed for each axis or overall (e.g. when using the distance in a joint relation) and whether this means that it has to be increased, decreases or simply changed in general.

For example, a **joint relation recognizer** requires that the left hand is placed over the shoulder, but the current user has placed it 20 cm lower. In this case, the recognizer reports that the pose of the left hand has to be changed according to its position. The position does not have to be increased or decreased in general, but it has to be changed for 20 cm in direction of the positive y-axis.

In a different example, an **angular movement recognizer** requires that the head is pitched downwards with at least 15 $^\circ/s$, but the user is moving the head in the opposite direction with 20 $^\circ/s$. Here, the recognizer reports that the angular speed needs to be changed. Again, no general increase or decrease, but -35 $^\circ/s$ around the x-axis.

For easier reporting the hints to the users, FUBI can construct **text messages** out of the correction hint's values. For the above examples, those would be "Please move left hand more up: 200 mm" and "Please turn head faster down: -35 $^\circ/s$". The information of the correction hints is also used for the **corrective arrows** of the automatically generated gesture symbols (see Figure 6.1c), which do not emphasize the motions performed by the virtual character, but actually visualize what the user has to do next to mimic the displayed gesture.

Combination recognizers report correction hints as well. At first, the hints include the state for which the recognizers failed. This can be the current state that has been interrupted, or the next state which currently cannot be reached as its recognizers are not fulfilled, yet. The rest of the recognition hint is set to the information provided by the first failed recognizer within that state.

In addition to the correction hint, combinations also log the user tracking data at the entering and leaving of each state. This information can be accessed to further investigate the current recognition progress. Com-

binations also have a separate method, that reports additional information about the current state of the recognition. This includes the index of the current state, the total number of states and whether the state is currently interrupted or in transition to the next state.

Last but not least, combinations support that the gesture designer includes meta information for each of the combination's states. The meta information of the current state can be requested during runtime. In FUBI's Unity integration, this is used to link each state to the frame index of the animation used as a gesture symbol for the combination. As soon as the user starts the combination gesture, the symbol exactly displays the part of the animation that has to be performed next.

6.1.4 Was My Interaction Successful?

Even when users already know how they can interact and what their actions would cause within an application, and they get support during the interaction, it can still happen that the interaction fails, e.g. because of an imprecise performance of the gesture, or because of distortions from the environment. Therefore, it is important to tell the users whether the interaction was successful, but in the case that something went wrong, it is also important to provide information on how to improve.

Feedback for successful interactions is usually quite straight-forward. For example, if an item in the GUI has been selected successfully, the selected item gets highlighted, optionally using an animation or the parallel playback of a sound clip. In the general case that the input gesture is linked with an arbitrary event in the application, first of all, the event gets triggered, but additionally, a corresponding gesture symbol can be highlighted, using an animation (see the symbol for action "Talk" in Figure 6.2) or the parallel playback of a sound clip if desired.

Feedback for failed interactions is usually more difficult, as the system may need to detect that an interaction attempt has been made in the first place. Furthermore, it needs to consider what went wrong and how this can be solved. However, FUBI's correction hints can again be used to provide such information as described in the preceding section.

Figure 6.2: Highlighting of a triggered onscreen symbol

6.2 Comparison of Gesture Visualizations

We already started to investigate different support mechanisms in more detail. Therefore, this section is first dedicated to the question "How can I interact?" as formulated in Section 6.1.2. In that section, I already mentioned the use of gesture symbols to visualize how an input gesture has to be performed by the user. In the following, I compare three visualization techniques for gestures symbols in an evaluation study as previously presented in [94].

Related work, as described in Section 2.6, only rarely compared multiple visualization techniques of full body interaction. Tang et al. [174] performed an informal study with their movement visualizations which indicated that the 3D versions of arrows and lines made it easier to perceive directions than the 2D versions, while both types still needed improvements regarding depth perception. Tang et al. further evaluated their advanced system [175] for comparing the wegde visualization, the multi-camera view, and a standard video view. In this second study, they found that combining the multi-camera-view with the wedge visualization resulted in the least errors in the users performances. Anderson et al. [4] showed that their training system resulted in better short-term retention scores than traditional video-based instructions. Walter et al. [184] investigated a public display setting. They concluded that more users executed the gesture of a symbol in a dedicated area instead of one surrounded by other content. Interrupting the current display and showing the symbol alone covering

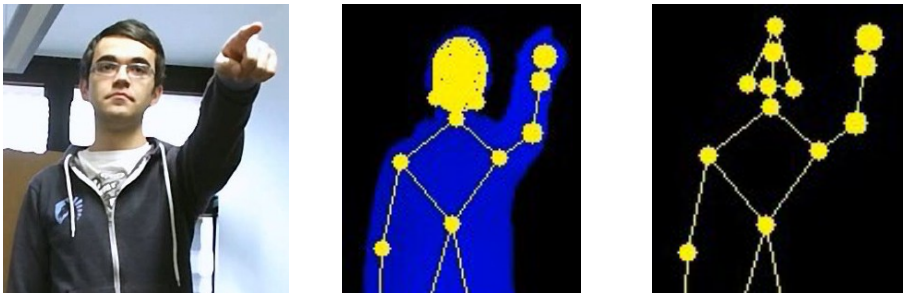
the full screen, made more users stop the interaction and leave instead. In general, designers of full body interaction apply a certain visualization technique according to their preferences (e.g. color videos [196] or virtual mannequins [145]), but they do not investigate further options and a comprehensive formal evaluation is still missing.

6.2.1 Visualizations

As indicated before, there are multiple options on how to visualize full body gestures. They have in common that a whole humanoid body or body part is displayed performing the gesture. The form of the humanoid body can vary between displaying the image of a real person (cf. [196, 4, 180]), abstracting it by its shape or skeleton (cf. [184]), illustrating it with a cartoon-like image such as a virtual character or a simple stick figure (cf. [145] and most commercial full body interaction games), or employing a non-human body such as a robot, an animal, or even other humanized objects. For static postures, it is enough to display a single image of the body in pose, optionally highlighting important body parts. For dynamic gestures, there are multiple options on how to visualize the gesture's motions. The simplest and most common way to visualize motion is to animate the image, i.e. playing it as a video, either with a frame rate high enough to cover the whole motion, or with a lower frame rate only depicting important states of a gesture. Another option is to include lines or arrows in the image or video to emphasize motion trajectories (cf. [4, 174] and most commercial full body interaction games). In this way, even more complex gesture shapes can again be visualized within a single image and might be perceived more easily. There are other, more artistic options to visualize motions. For example, motions can be visualized by changing the form of an object or showing a preparation pose (cf. animation principles "squash and stretch" and "Anticipation" [109]), using motion blur, quiver lines, or double takes, i.e. blending images of multiple motion stages into a single image (cf. [61]). However, those options are very rare in the literature as well as in commercial full body interaction games.

In the study described in the following section, I focused on three ways of

displaying an actual human body that is used to visualize the gesture. This is a common practice in scientific studies and makes it easy to create the gesture symbols, as one only needs to record a sample gesture performance. Nevertheless, there are still many options on how to visualize the gestures when using the recording of a depth sensor. We investigated three body visualization techniques which are depicted in Figure 6.3. In the "Color"



(a) Color (b) Shape (c) Skeleton

Figure 6.3: Pointing gesture visualized with three different techniques

technique (cf. Figure 6.3a), a gesture performance was captured on color image as done by Zafrulla et al. [196]. In the "Shape" technique (cf. Figure 6.3b), the actor was abstracted by only displaying a uni-colored shape enhanced with the his tracking skeleton and face mesh equal to the Traveller application of Section 4.3. This was further simplified in the "Skeleton" technique (cf. Figure 6.3c), in which only the Kinect's tracking skeleton with a simplified face was used. The latter might be closest to the schematic drawings used in some commercial full body interaction games, e.g. "Dance Central", "Kinect Adventures", however, the symbols did not involve manual design. For all three techniques, dynamic gestures were animated as a video with 25 frames per second, but no other enhancements, such as lines or arrows, were introduced. All visualizations had in common that they were generated automatically out of a user's recorded gesture performance, and they only differed in which part of the sensor information they presented.

Regarding the three visualizations, the hypothesis was that "Color" should make it the most easy for users to reproduce the gestures in an accurate way. The reason is that this technique represented the gestures as they

are seen in real-life on other people, although without stereoscopic vision. The other two techniques simplified the visualization and only displayed the information used by the tracking system. As those techniques therefore omitted presumably unnecessary details, we assumed that users were faster in starting to perform the gestures. Nevertheless, as the "Skeleton" technique completely abstracted from the actual human shape, it might be again more difficult to translate back to the actual body movement in comparison to the "Shape" technique.

Eighteen gestures plus one tutorial gesture were chosen for the study and are illustrated in Figure 6.4. The tutorial gesture (Figure 6.4a) and the first six gestures are depicted in the "Color" technique, the next six gestures in the "Shape" technique, and the last six gestures in the "Skeleton" technique. The symbols for dynamic gestures are represented in the table with a sequence of two to four key frames of the corresponding video, which, in the running system, were played with 25 frames per second.

Furthermore, gesture recognizer were defined in FUBI XML to recognize those gestures in a relatively unstrict manner (false positives were not an issue in this study). As the recognition only played a minor role in this study, I do not further describe the recognizers here, however, corresponding XML definitions can be found within Appendix A.2.

6.2.2 User Study

The experiment was arranged in a room of about 3 meters width and 6.5 meters depth. The participants were standing at a distance of about 2 meters in front of a 50 inch plasma display with a Microsoft Kinect for Windows 2 placed just below the screen in a horizontally centered position. The experimenter was sitting to the left of the participant and controlling the application running on the display via mouse and keyboard.

After a short introduction and a demographic questionnaire that also asked about the users' experiences with body gesture based interaction, the experimenter explained the participants their role in the study. At first, a timer was displayed on the screen and was counting down from five to zero. As soon as zero was reached, the first gesture symbol was displayed on the

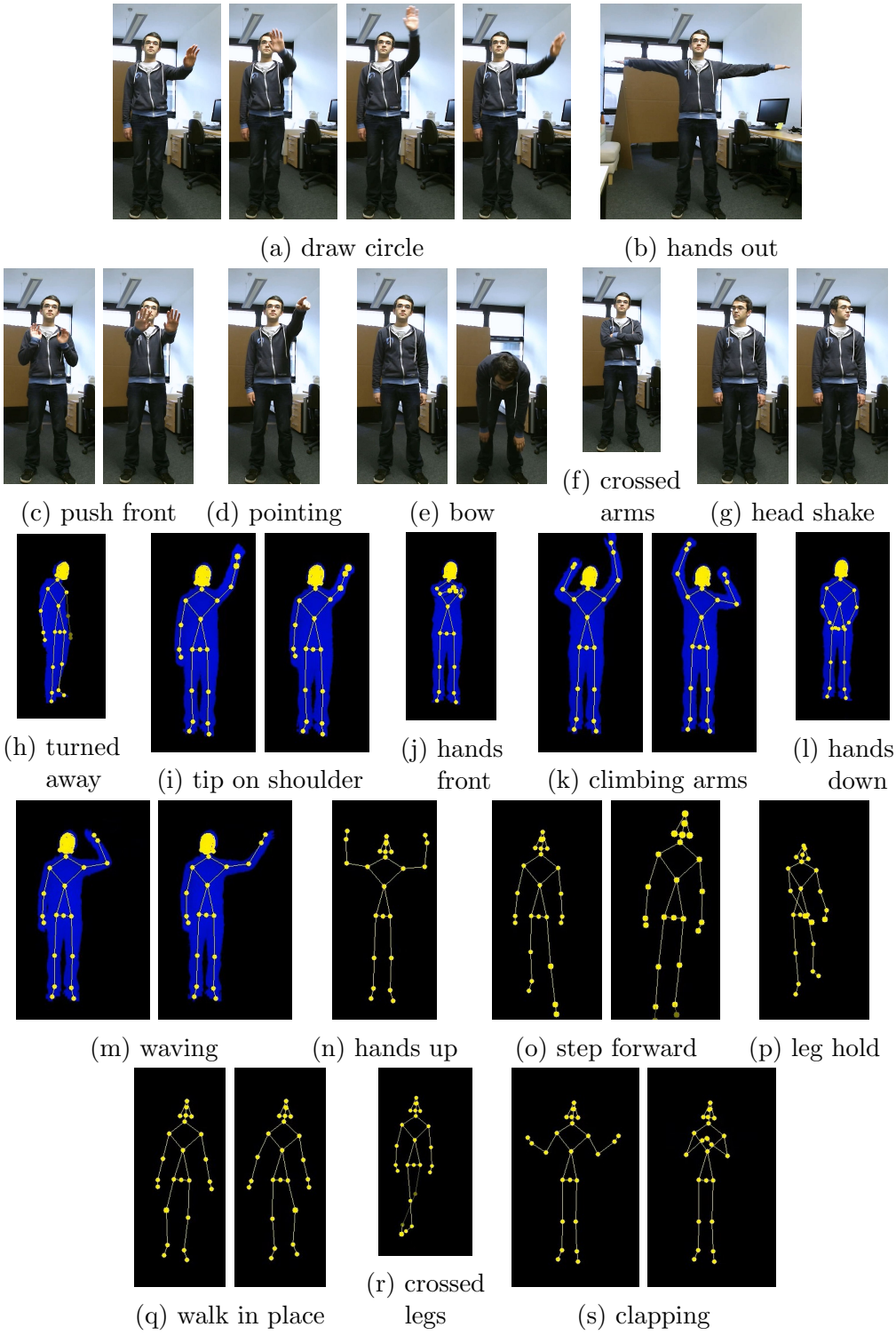


Figure 6.4: Symbols used in the gesture visualization study

screen. The participants should look at the symbol, and as soon as they understood how the gesture needs to be performed, they should immediately start performing the gesture. However, the gesture performance itself should be done as precise as possible without trying to be fast. No matter what happened, the gesture should always be completed. Furthermore, the participants were told that the animated gesture symbols were repeated infinitely, but they should start the gesture as soon as one iteration had been played and they understood how to perform it. As soon as the system recognized the gesture performance, the symbol was blended out and the count down timer started again for the next gesture. All participants took part in all conditions, and therefore saw all three types of visualizations, however, the order of visualizations was counterbalanced between the participants. For each visualization, the participants first saw the tutorial gesture a) and then six gestures of the gesture set, i.e. gestures b)–g) in the first visualization, gestures h)–m) in the second visualization, and gestures n)–s) in the third visualization. As can be seen in Figure 6.4, the symbols were alternating single images and animated videos corresponding to static and dynamic gestures. After the gestures of each visualization, the participants filled in a short intermediate questionnaire, which included six questions regarding the usability and intuitiveness of the visualization technique with alternating positive and negative wording, i.e. (translated from German):

1. I could easily recognize how the gestures should be performed.
2. The gestures' presentation was unnecessarily complex.
3. I could reproduce the gestures immediately.
4. I found it difficult to reproduce the gestures.
5. I found that the gestures' presentation was intuitive.
6. I found the gestures' presentation incomprehensible.

The questions were answered on a five-point Likert scale ranging from "strongly disagree" (1) to "strongly agree" (5). Further, the participants should name gestures that had been especially hard or easy to reproduce.

During the study, the program recorded the depth and color stream of

the Kinect as videos, and it recorded the user's tracking data in the FUBI recording XML format. It further measured the time from the display of the symbol to the recognition of the corresponding gesture, or alternatively to the experimenter pressing space bar on the keyboard. These timings were synchronized to the recorded video streams and stored in the ELAN [187] annotation format, while marking whether the gesture had been recognized automatically, or the experimenter had pressed space bar. These annotations were later enhanced manually with the time from the display of the symbol to the start of the gesture's preparation phase (the first movement of the participant introducing the gesture performance).

Eighteen participants took part in the study including two females. Their age ranged from 22 to 33 with a mean of 26.39 (SD = 2.79). All except for two were right-handed. The participants were recruited from our university campus and all had a computer science background. They stated themselves a medium experience with body gesture based interaction of 2.94 (SD = 1.06) on a scale from 1 (no experience) to 5 (daily usage).

6.2.3 Results

Regarding the responses of the questionnaires, we calculated an overall score for each visualization per participant. Therefore, we inverted the negatively formulated questions (2, 4, 6) with $x_{inv} = 6 - x$. Then we calculated the mean of the positively formulated questions and the inverted negative ones giving a rating of the visualization technique in the range of 1–5. A one-way repeated measures ANOVA indicated that there was a significant effect of the type of visualization on the overall score, with $F(2, 17) = 9.59$, $p < 0.001$, $\eta^2 = 0.36$. Post-hoc tests with Bonferroni correction further showed that the "Color" technique was rated significantly better than the other two techniques. "Color" reached an mean overall score of 4.78 (SD = 0.40) being significantly higher ($p < 0.01$) than "Shape" and "Skeleton" (r_{Color_Shape} = 0.65, $r_{Color\_Skeleton}$ = 0.68). However, there was no significant difference ($p > 0.05$) between the mean score of "Shape" which was 3.97 (SD = 0.99) to the one of "Skeleton" that reached 3.56 (SD = 1.20). To get further insights on the results of the questionnaire, we also compared its six items

using a MANOVA, and again, there was a significant effect of the type of visualization, with $F(12, 94) = 1.93$, $p < 0.05$ and Phillai's trace $V = 0.40$. In fact, Bonferroni corrected post-hoc tests, at first reported very similar results as the ANOVA of the overall score, with the mean of all six items being significantly better rated for "Color" than for "Skeleton" $p < 0.05$, but none of the items showed a significant difference between "Shape" and "Skeleton". However, for the comparison of "Color" and "Shape", only half of the items (1, 5, 6) was significantly better rated ($p < 0.05$) for "Color" than for "Shape", but the other half of the items (2, 3, 4) showed no significant differences ($p > 0.05$).

We compared the time it took the participants to start the gesture performance after the symbol had been displayed using the annotations mentioned in the preceding section. As the assumption of sphericity had been violated ($\chi^2(2) = 13.63$, $p < 0.05$), we calculated Greenhouse-Geisser corrected estimates, which were significant with $F(1.27, 21.61) = 5.01$, $p < 0.05$. Post-hoc tests revealed that participants were significantly faster in starting the gesture with the "Color" technique with an average of 1.60 seconds ($SD = 0.38$) than the other two techniques ($p_{Color\_Shape} < 0.01$, $r_{Color\_Shape} = 0.68$, $p_{Color\_Skeleton} < 0.05$, $r_{Color\_Skeleton} = 0.57$). Nevertheless, there was again no significant difference ($p > 0.05$) between "Shape" with 1.86 seconds ($SD = 0.42$) and "Skeleton" with 1.97 seconds ($SD = 0.72$).

Last but not least, we compared the number of wrongly performed gestures (="wrong") and the number of gestures rated as especially hard to reproduce (="hard"). For both, Friedman ANOVAs reported a significant effect ($p < 0.001$) of the type of visualization technique, $\chi^2_{wrong}(2) = 16.45$, $\chi^2_{hard}(2) = 12.81$. For Bonferroni correction, the post-hoc Wilcoxon tests are reported at a significance level of 0.0167. In both cases, only the comparison between "Color" and "Skeleton" was significant ($p < 0.0167$, $r_{wrong} = -0.58$, $r_{hard} = -0.55$), while the comparisons between "Shape" and the other two techniques were non-significant ($p > 0.0167$). "Color" had no wrongly performed gestures and an average of 0.22 ($SD = 0.43$) gestures were rated as hard to reproduce. "Shape" had an average of 0.33 ($SD = 0.49$) wrongly performed gestures and 0.78 ($SD = 0.73$) gestures were rated as hard to re-

produce. "Skeleton" had an average of 0.78 (SD = 0.55) wrongly performed gestures and 1.00 (SD = 0.59) gestures were rated as hard to reproduce.

6.2.4 Conclusion, Discussion, and Future Work

We presented three techniques for visualizing full body gestures, which were generated automatically by recording the gesture performances with a depth sensor. The techniques were tested in a study with 18 sample gestures preformed by 18 participants. During the study, videos of the participants were recorded and automatically pre-annotated using the FUBI real-time gesture recognition system. We found clear preferences for using color images of real persons to visualize full body gestures. This can be seen in the subjective participants' ratings as well as in the objectively measured time it took them to start performing the gesture and the number of wrongly performed gestures.

The reason that it was easier for the participants to learn the correct performance might be that the color images represented the gestures in a similar way they would be perceived in real life. According to the comments of the participants, the other more abstract gesture visualizations made it especially hard to recognize how the joints should be positioned in depth.

While many scientific approaches already use color images for gesture visualizations, the others should reconsider whether they have good reasons to use abstractions. Using recordings of a real person is common practice in research, as it allows to rapidly create gesture symbols without advanced design skills. However, this practice is almost never applied in commercial games, probably for aesthetic reasons. As we did not investigate the more synthetic visualizations that are commonly used in the gaming industry, the findings of our study do currently not allow clear recommendations for commercial games.

In a next step, the additional visualization option as mentioned in subsection 6.1.2 could be integrated to investigate a broader range of options, to better represent commercial full body interaction games, and to improve depth perception. Another option to improve the depth perception could be the use of a stereoscopic display, although this would be harder to setup

in real-life and it will probably only be an option for applications that are already using stereoscopy. Alternatively, the symbols could be displayed from multiple viewing angles [195]. Furthermore, other visualization options as mentioned in Section 2.6 could be used, e.g. schematic drawings which could make the perception easier while still being relatively abstract. However, this would require more manual post-processing or a completely manual creation by the interaction designer.

Chapter 7

Evaluations

In this chapter, I will present sample implementations and evaluations of the main contributions of this dissertation, which were described in the preceding chapters. I will first describe a sample implementation for full body avatar control in the context of culture-aware virtual characters in Section 7.1. Section 7.2 will be dedicated to freehand GUI interaction and evaluate text input with virtual onscreen keyboards. In the last two sections, I will present the implementations of the two user-defined gesture sets created in Chapter 4 and evaluate them regarding recognition accuracy and user experience. To implement the gestures, we mainly used FUBI's full body gesture recognition capabilities, but also added freehand GUI interaction in one case. During the integration of the gestures, we also added techniques for supporting the user during the interaction.

7.1 Evaluation of Full Body Avatar Control

An application with full body avatar control especially has the potential to invoke a strong feeling towards the non-verbal behavior of the virtual characters. Therefore, we used this approach in a cultural training scenario to test whether and how the users non-verbal behavior changed according to the culturally adapted non-verbal behavior of virtual characters that they

interacted with in [97]. Our approach distinguished from previous work by combining full body interaction with virtual characters simulating individualized interactive behaviors. System controlled characters responded to the users' avatar by dynamically adapting their behavior depending on their own assumed cultural background.

7.1.1 Culture-Related Non-Verbal Behaviors

Culture-related differences manifest themselves on more than one channel. While most work focused on behaviors that were usually expressed in a conscious manner, such as speech, we concentrated on nonverbal communicative behavior, which included gestures and postures, but also social navigation behaviors, such as interpersonal distance or body orientations. Hall [63] distinguished between four different distance zones: intimate, personal, social and public distance. For Northern Americans these zones were found as: intimate zone up to 0.45 m, personal zone from 0.45 m to 1.2 m, social zone from 1.2 m to 3.6 m and the public zone starts at 3.6 m.

The influence of culture on interpersonal distance was studied by various researchers. Sussman and Rosenfeld [171] for example studied the influence of culture, language and gender on conversational distance based on Hall's proximics theory. Their results were particularly strong distinguishing high- and low-contact cultures which was exemplified for the Venezuelan and Japanese cultures. Ferraro [47] found that the average conversational distance for European Americans was approximately 0.5 cm. For Latin Americans this distance could drop down to 0.35 m. Arabian cultures perceived the conversational distance to be as low as 0.22 m. Watson [185] noted that in high-contact cultures interpersonal distances were small, interlocutors faced each other directly, often touched one another and spoke in a low voice, whereas in low-contact cultures interpersonal distances were greater, interactants faced each other more indirectly, spoke in louder voices and touching was less usual. Hofstede [68] postulated that members of individualistic cultures such as the US were likely to stand out visually during interaction. Thus, their interpersonal distance should be rather high. Collectivistic cultures, vice versa, were physically very close with in-groups, but

reserved with out-groups. Assuming a conversation with friends or family members, the interpersonal distance should be smaller for collectivistic cultures compared to individualistic cultures.

7.1.2 Evaluation of Full Body Avatar Control and Different Interpersonal Distance Behaviors

To test our approach of full body avatar control in combination with our approach to culture-specific agent parametrization, we conducted a first study with 26 volunteers to investigate (1) how intuitive full body interaction is for users and (2) how they respond to the agents' culture-specific behaviors, while we hypothesize that users prefer agent behavior designed to resemble their own cultural background. For the latter, we focused on culture-specific social navigation behavior and in particular interpersonal distance behavior. Our work extends studies by Bailenson et al. [7] who investigated the influence of gender on human proxemics behaviors in virtual environments by studying culture as a novel variable.

Figure 7.1: Virtual characters with prototypical individualistic (left) and collectivistic (right) spatial behavior

To this end, we created three kinds of characters, showing different culture-related interactive behaviors: (1) Neutral virtual agents showing mediate spatial behavior (65 cm – 1.2 m). (2) Prototypical collectivistic agents that keep a closer personal distance (35 cm–85 cm). (3) Prototypical individualistic agents that have higher interpersonal distance constraints (95 cm – 1.55 m). Figure 7.1 exemplifies two groups of virtual characters

that show prototypical individualistic (left) and prototypical collectivistic (right) spatial behavior.

Evaluation Setup and Procedure

Users controlled an avatar following the approach described earlier. The avatars were shown from a third person perspective and replicated the users' behavior in real-time. Figure 7.2 depicts our setup with the user in front of a screen and a Microsoft Kinect for Xbox 360 placed right, below it. The screen displays our virtual beer garden scenario where the user avatar (woman shown from the back) and two other virtual agents are talking to each other.

Figure 7.2: Setup for our full body avatar control

The evaluation started with a brief introduction, and after that the participants were allowed to test the full body avatar control. Then, we explained the social setting to the participants and told them to follow a simple script. The user was supposed to have an appointment with two friends in the Virtual Beergarden. When his or her avatar shows up in the Virtual Beergarden, the friends are already there. As a first step in the script, the user had to perform a waving gesture for greeting the agents from distance and gaining their attention. Once the gesture was recognized by the system, the virtual characters waved back and verbally greeted the participant. As a next step, the user was requested to approach the group

of virtual characters in order to join them for a conversation. After a few sentences spoken by the virtual characters, they stated their farewell and walked away, which finished the scenario.

Participants had to perform three test runs to ensure that they got used to the script and were not distracted by factors other than the virtual characters' behaviors. For these practice interactions, we used the neutral agents. Afterwards, the actual evaluation study started. Participants had to run through the same script for two more times with the virtual characters that had increased or decreased distance zones, shown in random order. To avoid any bias caused by gender, all character groups consisted of one male and one female character. All characters had a Western appearance with varying clothing, hair style and eye color randomly generated at each start of the application.

After each interaction in the evaluation study, we asked the participants to fill in a questionnaire addressing two major aspects: (1) the effectance of the interaction and (2) the impression the characters conveyed. All answers had to be given on a 7-point Likert scale with optional comments. To evaluate the effectance, we asked participants whether they felt that they had accomplished their task and whether they understood why the characters were behaving as they did. To evaluate participants' impression of the characters, they had to rate four adjectives: natural, intrusive, open and likeable. After completing the interaction with both groups of characters, participants were asked to fill in a general questionnaire. Besides some demographical questions, we asked how intuitive the interaction with the virtual characters via Kinect was perceived, and asked them to comment on the observed differences between the two virtual character groups. We additionally took video recordings of all interactions.

Evaluation Results

We recruited 26 volunteers from our university campus to participate in the study. The mean age of the participates was 28.12 (SD = 5.9), with age ranging from 23 to over 60.

First of all, we investigated whether participants found the interaction

with the virtual characters using the Kinect intuitive. A t-test for one sample, testing whether the user rating was significantly above the neutral value of 4, revealed, that our participants thought the interaction was easy with a mean value of 6.231; $t(25) = 9.959$, $p < 0.0005$. There was also evidence that our users enjoyed the interaction with the Kinect: Most participants started spontaneously playing with the system and tested the navigation and gesture imitation even before we introduced the system to them.

For the effectance ratings, we did not get significant differences between the two conditions (collectivistic and individualistic characters), but we applied t-tests for one sample that revealed that all ratings were significantly above the neutral value of 4. The participants thought that they had accomplished their task while interacting with the prototypical collectivistic characters ($M = 6.65$; $t(25) = 18.158$, $p < 0.0005$) and with the individualistic characters ($M = 6.73$; $t(25) = 26.100$, $p < 0.0005$). The participants further indicated, that they understood the characters' reactions with the prototypical collectivistic characters ($M = 6.19$; $t(25) = 10.953$, $p < 0.0005$) and with the individualistic characters ($M = 6.12$; $t(25) = 8.071$, $p < 0.0005$). These equally positive results in both conditions may be due to the fact that participants had the chance to get familiar with the system during the training phase.

To investigate whether participants rated the characters differently in the two conditions, we applied t-test for paired samples. For the obtrusiveness dimension, we found significant differences ($t(25) = 3.729$, $p < 0.001$). Thus, participants from Germany perceived virtual characters that had decreased interpersonal distance zones as significantly more obtrusive ($M = 4.08$) than virtual characters with increased zones ($M = 2.42$). We take this as evidence that users consider social distance behaviors that reflect their own culture as more appropriate than social distance behavior that do not. This result is in line with the literature according to which violations of interpersonal distance by stepping too close are described as being pushy or obtrusive, as described in the beginning of this section. We did not achieve significance for the other dimensions. We assume that the participants did not connect these values to the spatial behavior of the characters.

As Bailenson et al. [7], we did not inform the participants that we were interested in proxemics behaviors and that these behaviors were varied in the two conditions. Also in the questionnaire, we did not explicitly refer to the agents' proxemics behavior, but only asked the participants whether they found the agents' behavior plausible. Nevertheless, the optional comments of the questionnaires revealed that most participants (16 out of 26) noticed that the agents behavior varied in their interpersonal distance behavior, while 10 participants did not observe any differences.

From the video recordings, we moreover noticed that some participants felt violated in their interpersonal distance by the group with decreased distance and e.g. continuously tried to retreat by stepping back. Therefore, we assume that interpersonal distance behavior between virtual characters and user avatars is perceived in a similar manner as human proximity behavior.

7.1.3 Conclusion and Discussion

In this section, I investigated a full body avatar control that brings together full body interaction in physical space that supports intuitive behavior (gestures and movements) with the social nature of the virtual agents' culture-specific behavior. Our interactive system revealed that full body interaction can enrich applications with virtual characters and users found the control of an avatar by their body movement intuitive. In our first evaluation study, we focused on interpersonal distance behavior. The study indicated that human users noticed cultural differences regarding the distance behavior reflected by the agents and responded to them in a promising way. We thus saw great potential of device-free interfaces for cultural training scenarios in particular. In general, the avatar control could be a good mechanism to integrate the user into a virtual world. Users could interact in a natural manner and responded to virtual characters spontaneously without being disturbed by obtrusive interaction devices. Nevertheless, avatar control alone is usually not expressive enough to cover all intended interaction possibility. Therefore, we already integrated additional interaction based on actual gesture recognition with the waving gesture.

7.2 Evaluation of Freehand GUI Interaction

In Section 5.3.1, I described multiple approaches for implementing virtual keyboard text input as an example of freehand GUI interaction. In the following section, I will evaluate these approaches in two different user studies.

7.2.1 Initial Evaluation of Standard Layout Virtual Keyboards

In a first evaluation study, we compared the four different text input methods using a standard layout virtual keyboard.

Setup and Procedure

The main application scenario for our approach is when already interacting with a depth sensor while standing in front of a more or less large screen and having the need to write a short amount of text. Figure 7.3 displays the setup as used in our evaluation study. Users were standing about 2 meters in front of a 50 inch plasma display with a Kinect placed just below the screen in a horizontally centered position. For rendering the graphical interface, we used our Horde3D GameEngine for having full control over the design and behavior of the interface.

Figure 7.3: Study setup for freehand text input

After a short introduction, users were encouraged to move the cursor on the screen to get used to the cursor control mechanism, and the mapping between tracking and screen coordinates was calibrated for the users being able to reach the whole screen with the cursor in a comfortable way (top left corner with right hand slightly left of the head, bottom right corner with right hand in hip height and, dependent on the arm length, about 40 cm outward). For *dir-push* that uses spherical coordinates, the calibration was done separately just before its execution.

We applied the "within subjects" design, i.e., all participants used every input method. The first task within each method was to enter one test line for getting used to the interaction. Thereafter, participants had to enter four different lines for which the input times were measured. Since we focus on scenarios where users tend to provide rather brief text input, we only used text lines with an entire length between 8 to 13 characters, each to be finished with a return key and alternating one line with one word and one line with two words. The currently requested line was displayed at the top of the screen and the entered characters appeared in a text field directly below this line using a different font color and decoration in order to avoid that the users mixed them up. The chosen text lines covered all characters included in the keyboard layout, examples are "ein taxi" ("a taxi"), "machtkampf" ("struggle for power"), or "eisenring" ("iron ring"). This was important as letters at the border of the keyboard are usually less easy to write using the input methods that include a pushing gesture with the cursor controlling hand. Nevertheless, we also tried to achieve a frequency per character that is roughly oriented at the distribution in the German language, e.g. with "e", "n", and "i" being the most frequent characters in the texts.

The participants were requested to write the lines fast, but with at least errors as possible. They were told to correct wrongly entered letters immediately. The order, in which the different input methods were encountered, was counterbalanced between the participants, to compensate for the influence of learning effects. However, the order of the different text lines was the same for each participant. In consequence, the same participant enters different text lines for each text input method, e.g. text lines x using

method a, and text lines y using method b, but the next participant might enter text lines x using method b, and text lines y using method a.

During typing, we logged the time it took users to write each line of text. The writing speed was then calculated in words per minute (WPM, one word = 5 characters) according to the formula by Shoemaker et al. [161]: $speed = 60 \times (|L| - 1)/(5 \times T)$, where L is the text line and T is the completion time in seconds.

After writing the four lines of one condition, the participants filled in a questionnaire based on the System Usability Scale (SUS) developed by Brooke [20]. The SUS consists of ten items with alternating positive and negative statements to be rated on a five point Likert scale. The ten statements of the SUS were translated to German and the wording was slightly modified to match our setting, e.g. "system" was replaced with "text input method". After the last text input method, the participants filled in an additional questionnaire for demographic data, questions related to their experience with keyboard typing and Kinect, and a question for their preferred text input methods, with multi-selection allowed.

Participants

We recruited 22 volunteers (6 females) from our university campus to participate in the study. The mean age of the participants was 28.36 (SD = 5.18), with age ranging from 22 to 42. All participants were right-handed and used to typing on a QWERTZ keyboard. They stated themselves a good typing experience with a mean value of 3.32 (SD = 0.65) and minimal value of 2 on a scale from 0 (no experience) to 4 (touch typing using all fingers). Thus, we can assume that all participants were used to the arrangement of the letters on our virtual keyboard. All participants were novice to the text input methods with the Kinect that were used in our setup, but 18 participants stated that they had played with or used the Kinect in any other way before at least once. In this experiment, each participant wrote 144 characters using the four different methods each of them taking about 1–3 minutes.

Results

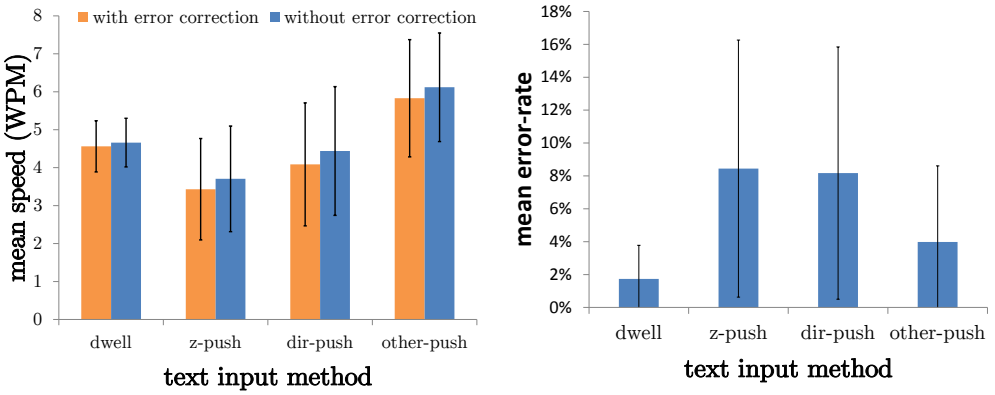


Figure 7.4: Average speed and error rates for standard layout keyboards

The average speed values and their standard deviations are depicted in Figure 7.4 on the left-hand side. As we removed the possibility to correct errors in the other two studies, we depict the speed with (orange) and without (blue) taking the times for error correction into account. You may note that the correction times only resulted in slightly lower writing speeds. To stay comparable to the other studies we will report our findings for the speeds without error correction in the following, however, they have also been validated using the speeds with error correction. We applied a one-way repeated measures ANOVA that confirmed that speed was significantly affected by the type of input method, $F(3, 21) = 33.26$, $p < 0.01$, $\eta^2 = 0.61$. In particular, the fastest input method was *other-push*. An average speed of 6.12 WPM ($SD = 1.43$) was reached, being significantly above all others ($p < 0.01$, $r_{dwell} = 0.80$, $r_{z\text{-}push} = 0.92$, $r_{dir\text{-}push} = 0.75$) taking the adjustment of the Bonferroni method into account. The second fastest method was dwell with 4.66 WPM on average ($SD = 0.64$). This value was significantly above the one achieved by *z-push* ($p < 0.05$, $r = 0.69$), but its speed was not significantly different from *dir-push* ($p > 0.05$). We therefore assume that *dir-push* might overtake dwell after a somewhat longer training phase. The third fastest method was *dir-push* with 4.44 WPM ($SD = 1.70$) that was also significantly above *z-push* ($p < 0.05$, $r = 0.60$) that reached the last place with an average speed of 3.71 WPM ($SD = 1.39$). One may note that only

dwell has a relatively low standard deviation while the others have a greater one. This is caused by the fact that most participants were immediately able to write quite confidently with this method, while there was a definitely greater difference between participants in the other methods.

The error rates, visualized in the bar chart in Figure 7.4 on the right-hand side, showed a different ranking, but again, we found significant differences between the four input methods. As parts of that data were non-normally distributed[1], we applied a Friedman's ANOVA that indicated that the error rates were significantly affected by the input method, $\chi^2(3) = 17.79$, $p < 0.001$. Wilcoxon tests were used to follow up this finding. All effects are reported at a 0.0125 level of significance for Bonferroni correction. The method with the lowest error rate was *dwell* with an average error rate of 1.74% ($SD = 2.04\%$), being significantly below the two approaches that used a pushing gesture with the same hand that controls the cursor ($p < 0.0025$, $r_{z\text{-}push} = -0.39$, $r_{dir\text{-}push} = -0.34$). The error rate of this method was not significantly different to the error rate of *other-push* ($p > 0.0125$) that reached an average error rate of 3.98% ($SD = 4.63\%$). However, other-push was again significantly below *dir-push* ($p < 0.0125$, $r = 0.26$) and *z-push* ($p < 0.0125$, $r = 0.27$). *Dir-push* reached an average error rate of 8.17% ($SD = 7.67\%$) and was not significantly different to *z-push* ($p > 0.0125$) that had an average error rate of 8.44% ($SD = 7.82\%$). The high values for the standard deviation of the error rates for all text input methods indicate that there were big differences between the participants. While for each text input method always participants existed that completed the whole task without any errors, others had serious problems with this new way of typing and one participant even reached an error rate of 36.1% with z-push.

By adding up and normalizing the user responses to the ten items of the System Usability Scale, we get a usability score that ranges from 0 (poorest usability) to 100 (perfect usability). We calculated this score for the user responses to our four text input methods and compared them again with a one-way repeated measures ANOVA. The results indicate that the text input method significantly influenced the usability score, with $F(3,$

[1] This is normal for error rates because in the ideal case, they are very close to zero.

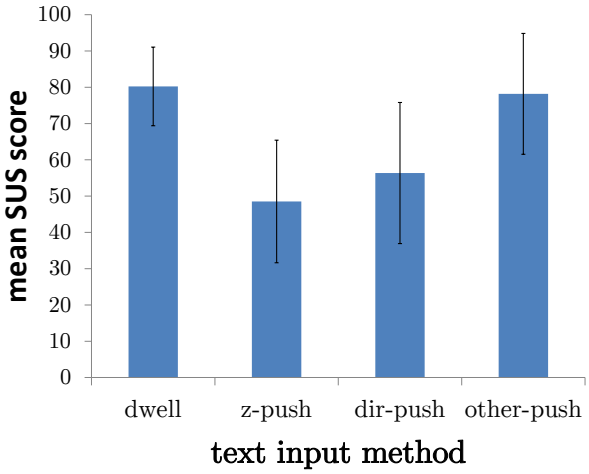


Figure 7.5: Average SUS score for standard layout keyboards

21) = 27.17, $p < 0.01$, $\eta^2 = 0.56$. Figure 7.5 contains the average SUS scores and their standard deviations visualized in a bar chart. The two best rated methods were *dwell* with an average score of 80.23 (SD = 10.83) and *other-push* with an average score of 78.18 (SD = 16.66), but there was no significant difference between them ($p > 0.05$). However, they were both significantly higher rated than the other two methods ($p < 0.01$, $r_{dwell+z\text{-}push}$ = 0.88, $r_{dwell+dir\text{-}push}$ = 0.76, $r_{other\text{-}push+z\text{-}push}$ = 0.79, $r_{other\text{-}push+dir\text{-}push}$ = 0.72). The third best rated method was *dir-push* with an average score of 56.36 (SD = 19.44), but it had no significant difference ($p >$0.05) to *z-push* that gained an average score of 48.52 (SD = 16.88).

We also had a closer look at the ten items of the usability questionnaire in particular and applied a one-way repeated measures ANOVA on each of them individually mainly looking for cases were the result differs from the one of the overall SUS score. In fact, we did not find any significant differences between *dwell* and *other-push* for any of the ten items again ($p >$ 0.05). In parallel, all items had again significant differences between *dwell* and *z-push* ($p < 0.05$). Between *other-push* and *z-push* it looked nearly the same, only for item 4 there was no significance indicated ($p > 0.05$), but it may be due to the fact that the wording of item 4 ("I think that I would need support to be able to use this text input method") did still not suit perfectly for the text input task, although it was already slightly adapted

from the original SUS version. In contrast to the overall SUS score, item 1 indicates a significant difference between *z-push* with an average rating of 1.05 (SD = 1.09) and *dir-push* with an average rating of 1.82 (SD = 1.33), $p < 0.05$, $r = 0.57$. Therefore the participants would prefer to use *dir-push* frequently than using *z-push* frequently.

In the final questionnaire that users had to fill in at the end of the study, 77.27% of the participants selected other-push as one of their favorites, 54.55% dwell, 13.64% dir-push, and none selected z-push.

7.2.2 Longitudinal Evaluation of Virtual Keyboards with Multiple Layouts

In a second evaluation study we investigated how the performance and usability ratings would develop over time. We did not consider *z-push* as it showed the worst performance in the previous section and the similar *dir-push* method achieved better results. In addition, we added the methods we developed for the two new keyboard layouts shown in Figure 5.10.

Setup and Procedure

The setup in the second study was the same as in the previous study described in Section 7.2.1, but we slightly changed the procedure of the study. In particular, we removed the backspace key, so that users were not able to correct the text anymore. Instead the system refused the input of wrong characters and responded to them with an error sound. This approach is equal to the procedure used by Shoemaker et al. [161].

The participants ran through the study for five sessions within one week. They had to fill in the SUS questionnaires and the question about their preferred method only for the first and last session. The order in which each participant encountered the different methods was counterbalanced between the different sessions. After the last session, the users additionally wrote four sentences using the dwell based method, but this time with a reduced dwell time of 0.9 seconds (abbreviation: *fast-dwell*). This was to

test the error rate of more experienced users with adapted dwell time, and therefore probably higher performance.

Participants

Five male participants took part in the study consisting of five sessions within one week. All of them had taken part in the first evaluation study. Thus, we could assume a certain amount of experience with the standard layout methods. The mean age was 29.40 (SD = 4.22) ranging from 24 to 35. All participants were right-handed and stated themselves a good typing experience of 3.40 (SD = 0.55). They indicated a medium Kinect experience with a mean of 2.40 (SD = 0.89). In particular, four of them reported multiple, but not regular usage of the Kinect, and the remaining one indicated practically daily usage. None of them was involved in the development of the text entry methods in any way. Overall, each participant wrote about 900 characters with the virtual keyboards. Testing one method during a single session again took about 1–3 minutes.

Results

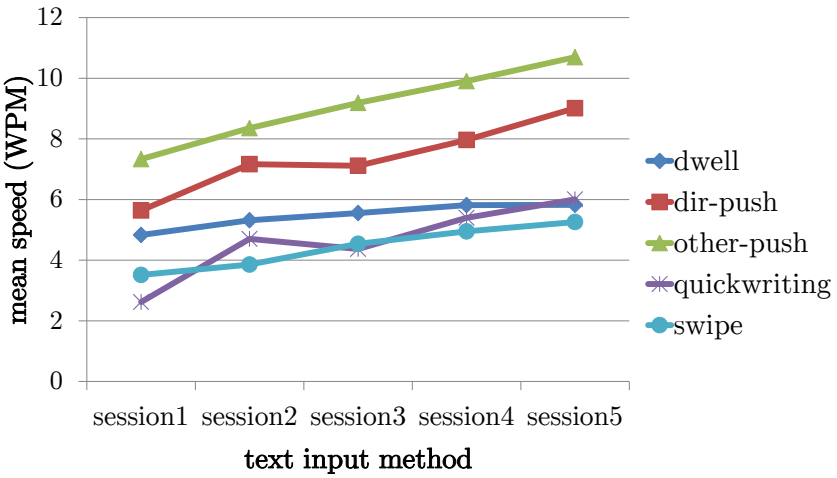


Figure 7.6: Writing speed learning curves for multiple layout keyboards

Figure 7.6 displays the learning curves for the writing speed within the five sessions. Already in the first session, the users achieved a higher writing

speed than in the previous study for the known methods, probably due to their prior experience. We applied a factorial repeated-measures ANOVA for Day x Method that revealed a significant ($p < 0.001$) effect of Day ($F(4, 16) = 26.13$), Method ($F(4, 16) = 42.32$) and the interaction between both ($F(4, 16) = 3.30$).

In particular, Bonferroni corrected pairwise comparisons showed significantly faster speeds ($p < 0.05$) on day 5 and 4 than on day 1, and on day 5 than on day 3. Furthermore, *other-push* is significantly faster ($p < 0.05$) than all other methods expect for *dir-push*, while *quikwriting* is significantly slower ($p < 0.05$) than all others except for *swipe*. Regarding the interaction between Day and Method, one interesting result of the contrasts is that there was a significant difference ($p < 0.05$) between day 1 and 5 comparing the speeds of *dwell* and all other methods ($F_{dir\text{-}push}(4, 1) = 55.69$, $F_{other\text{-}push}(4, 1) = 18.17$, $F_{swipe}(4, 1) = 9.73$, $F_{quikwriting}(4, 1) = 11.98$). Therefore, the writing speeds of all other methods improved significantly more than the one of *dwell*, which does not change much further from the third session on.

Overall, *other-push* and *dir-push* showed the greatest learning effects and it seems that they might still profit from further training sessions. Interesting is the development of the two new input methods: They both start at a relatively low level of ca. 3 WPM in the first session, being clearly below *dwell*. However, *swipe* gets close to *dwell* in the last session, and *quikwriting* is even able to outperform *dwell* slightly then, what might be further enforced after additional training. For *fast-dwell* the users reached an average speed of 7.03 WPM. This speed is again above the speed of *swipe* and *quikwriting*, but still clearly below that of *dir-push* and *other-push*.

The average error rate of *fast-dwell* was with 1.4% a bit higher than with the longer dwell time, but still clearly below the other methods. The learning curves for the error rate of the other five methods are displayed in Figure 7.7. Overall, the clear winner is *dwell* that stays at an average error rate of 0.5% after the second session, while all other methods have their minimal average error rate at around 5%. To confirm our results, we applied a factorial repeated-measures ANOVA for Day x Method that

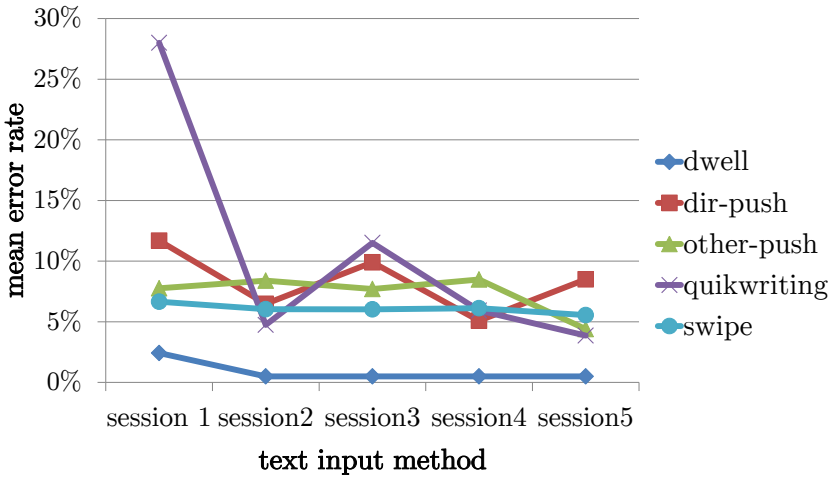


Figure 7.7: Error rate learning curves for multiple layout keyboards

revealed a significant ($p < 0.01$) effect of Day ($F(4, 16) = 5.36$), Method ($F(4, 16) = 5.53$) and the interaction between both ($F(4, 16) = 2.86$). The contrasts further indicate a significant difference ($p < 0.05$) for day 1 compared to day 2 ($F(1, 4) = 11.88$), day 3 ($F(1, 4) = 7.77$ and day 4 ($F(1, 4) = 19.71$ confirming an overall learning effect. Regarding the contrasts of the methods, *dwell* achieved significantly ($p < 0.05$) lower error rates than all other methods ($F_{dir\text{-}push}(1, 4) = 15.49$, $F_{other\text{-}push}(1, 4) = 7.98$, $F_{swipe}(1, 4) = 10.44$, $F_{quikwriting}(1, 4) = 12.64$). However, the contrasts for the interaction between Day and Method as well as the Bonferroni corrected pairwise comparisons of both did not provide any further findings.

Figure 7.8 depicts the SUS ratings the users gave to the five methods after the first and last session, and in addition, the SUS rating for the *dwell* method with reduced dwell time that was applied after the last session. *Dwell* and *other-push* were the two best rated methods with very similar ratings. A factorial repeated-measures ANOVA for Day x Method revealed a significant ($p < 0.01$) effect for the used Method ($F(1, 4) = 8.75$), but no effect ($p > 0.05$) of Day or the interaction between both. Bonferroni corrected pairwise comparisons further indicate a significantly ($p < 0.05$) higher SUS rating for *other-push* compared to *quikwriting*, but no further significances. It looks as *dir-push* and especially *quikwriting* were rated better after the last session, but this difference was non-significant. *Fast-*

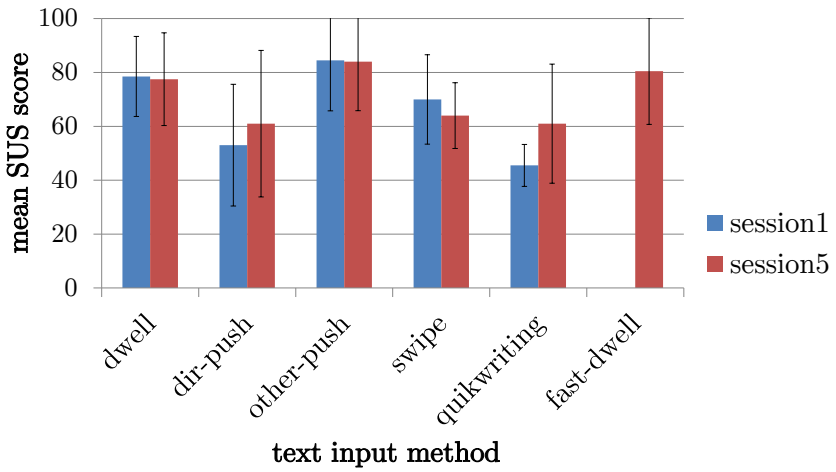


Figure 7.8: SUS ratings for first and last session of the longitudinal study

dwell was rated similar as *dwell*. Thus the reduced dwell time had no great effect on the rating.

The final questionnaires had similar results: After the first session two users chose *swipe* and all chose *other-push* as their favorites, but after the last session, two chose *dwell* and again all chose *other-push*. In addition, we asked the participants whether they would have needed more time to learn the new key arrangements. No one indicated this for *swipe*, but three would have liked more training for *quikwriting*. Before presenting them the *fast-dwell* method, we further asked the participants whether they would have liked a shorter dwell time. Three agreed, one was unsure and one participant disagreed.

7.2.3 Discussion of Virtual Keyboard Based Text Input

The speed of all our text input methods was considerably higher than the 1.83 WPM measured for the text entry speed using the Microsoft Kinect Xbox interface [71]. Their performances are also similar to the ones Ren et al. [148] measured. However, we did not get better results with our non-standard layouts, but our fastest method, *other-push* with 10.7 WPM after five sessions, outperformed their fastest method, *Reach* on a dual-circle keyboard layout that reached 8.57 WPM after five sessions. One

might argue that such a comparison is not quite fair as Ren et al. require the users to correct errors, however, our method should still be faster when adding error correction as it achieved 25% higher speeds, but only had an error rate of 4.4%. An approximation for the speeds with and without error correction can be seen in Figure 7.4.

All of the virtual keyboard based approaches can compete with some of the approaches using a hand-held device or marked gloves, e.g. 6.32 WPM with a gamepad [71], 6.5 WPM using a webcam and a colored glove [132], and 3.7 WPM with the Nintendo Wii Remote accelerometer [81]. Nevertheless, there also exist approaches with hand-held or worn devices that clearly outperform our Kinect based ones, e.g. 14.5 to 18.9 WPM with the IR tracking of the Nintendo Wii Remote [161] or 11.63 WPM with an OptiTrack motion capture system and marked gloves [117].

Unlike Ren et al. [148], but in line with Shoemaker et al. [161] and Markussen et al. [117], we obtained better results for the methods that were based on a standard layout keyboard both in terms of speed and usability. However, the measures for *quikwriting* are similar to those of the most equivalent approach by Jones et al. [81], and both *quikwriting* and *swipe* can compete with all other keyboard-based methods except for *dwell* in terms of error rate.

In particular, *other-push* seems to be the most promising approach. It outperformed all other methods in terms of speed, learning curve (together with *dir-push*) and perceived usability (together with *dwell*). We believe the main advantage of other-push lies in the fact that the movement to the target and its selection can be done in parallel saving time and increasing the simplicity of interaction. This made it possible that users reached an average writing speed of 10.7 WPM after five sessions.

However, *dwell* had the lowest error rate (around 2% in first sessions and a constant average of 0.5% in sessions 2–5 of the longitudinal study) and is thus immediately easy to use with few errors. The drawback is that it has a quickly reachable border of performance: the dwell time itself. For expert users, the dwell time can be reduced to achieve a higher writing speed. Nevertheless, this would need further usability investigations with

other text input scenarios and it does not seem that a reduced dwell time can outperform the keyboard based methods using a pushing gesture.

To explain that further: Users reached a writing speed of close to 6 WPM after five sessions with a dwell time of 1.2 seconds. This means, they overall needed around 2 seconds to enter one character, or 0.8 seconds to travel from one character to the other between the dwell phases (we also measured an equal travel time for *fast-dwell*). In comparison: *Other-push* gained 10.7 WPM, which is equal to 1.12 seconds per character. Therefore we would probably need to reduce the dwell time to around 0.3 seconds to get similar results in a dwell based approach. While this might be possible for gaze based interaction of expert users with an eye tracker, this would emphasize the Midas touch problem in freehand interaction to a degree that makes the system completely unusable. All other methods than *dwell* do not seem to need adaptations for expert users, but are more difficult to perform as a novice user.

7.2.4 Conclusion and Implications for General Freehand GUI Interaction

I presented a wide range of methods to implement character-wise freehand text input along with a systematic comparison and evaluation. Four methods transferred the point-and-click paradigm of current personal computers to a text input method using a standard layout virtual keyboard. Two methods used a different keyboard layout and a different key selection mechanism. In two studies, we evaluated these methods to get information about the possible performance and usability. Overall, I would recommend a dwell based virtual keyboard approach when text input is only rarely needed and no user training is foreseen, e.g. when entering an avatar's name at the beginning of a game. When text input is used more frequently, e.g. for inputting short search queries as in the Bing search of the Micorosft Xbox Kinect interface, it should be worthwhile to introduce a method where cursor movement and item selection are implemented as separated actions to enable dwell-free two-hand interaction as in our *other-push* approach. Although this will probably introduce more typing errors, the higher writing

speed should more than compensate this disadvantage after few training sessions, and the writing speed should still be significantly higher even when it is decreased by the users having to correct more errors. I would currently also recommend using a standard layout for virtual keyboards.

Similar recommendations can be implied for freehand GUI interaction in general. As long as there is infrequent input needed, e.g. the GUI is only displayed during pauses or for changing settings during a full body interaction game, the best solution will still be the well-known dwell based approach. However, when input is needed more frequently, e.g. the GUI interaction is part of the main gameplay, it is worthwhile to introduce other item selection methods. A method that includes a secondary body part as our *other-push* method is the fastest solution that we found when needing relatively precise input as with virtual keyboards. When there are only few options available as in a dialogue menu, it can be practical to use one of the methods that employ pushing gestures with the cursor controlling hand. One possibility on how to implement such a method with few options is the "button mode" presented in Section 3.1. As the GUI items are usually not very dense in this case, the cursor control can use a "snapping" mechanism, i.e. the cursor gets attracted to an item as soon as it is below a certain distance which is implemented in FUBI's Unity integration. Another approach with pushing gestures of the cursor controlling hand that profits from the low number of available options is the swipe menu that will be described in Section 7.4.2.

7.3 Evaluation of Gestures for Controlling Humanoid Robots

In [136], we implemented and tested a prototypical system for the control of a humanoid robot with the user-defined gesture set created in Section 4.2. Therefore, we defined a subset of the gesture candidates in FUBI's XML-based language. In particular, we used the candidates proposed by the NT users as shown in Figure 4.3. Considering the implications for gesture recognition, we eliminated any duplicate or similar gestures in the

set; for example, there were two NT gesture candidates for the action Move forward, one deictic and one iconic gesture, on the other hand, there were only iconic gestures for Move backward/left/right. Therefore, we chose the iconic gesture for the Move forward action to stay consistent with the other Move actions. Furthermore, there were nearly identical NT gesture candidates for the actions Slow down (NT (b)) and Sit down (NT (a)), thus, we decided to use the NT (b) of the Sit down action. The gestures we chose for all actions can be seen in the second column of Table 7.1. Moreover, gestures that include repeated movements within the stroke phase were implemented to be classified from a single performance of that movement.

7.3.1 Implementation of Gesture Candidates

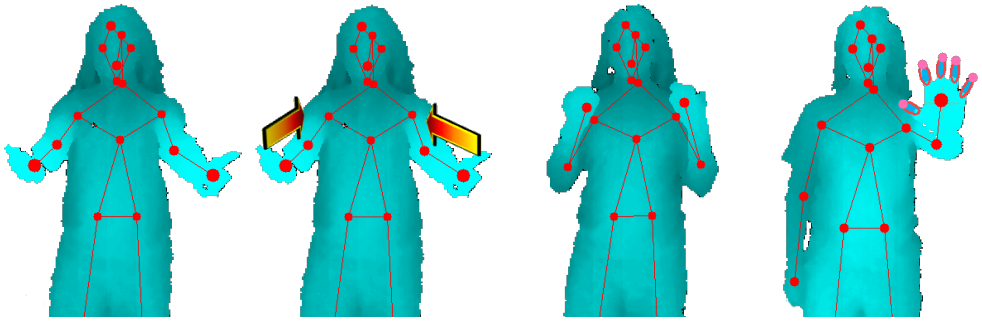
According to the timings displayed in Figure 4.3 and the video recordings of the gesture performances, we tried to implement the gesture classifications as close as possible to how the users actually performed them for each action. However, there were some restrictions of the depth sensor based tracking system that we needed to take into account as well.

The most obvious modification was for the action Sit down, for which we implemented the gesture candidate in which the user is actually sitting down on the ground. However, when a user is sitting down on the ground, the tracking becomes very unstable, looses several joints, and sometimes even fails completely. Therefore, we chose to modify this motion, so that the user should not completely sit down on the ground, but only adopt to a squatting position with the knees bent about 90 degrees. The recognizer for this gesture was accordingly looking at the orientation of both knee joints, and waiting for a high rotation around the x-axis that needs to be kept for at least 0.5 seconds.

The recognizers for the actions Move forward/backward/right/left, were all implemented in a similar way. At first, they all require at least a short period with no body movement to avoid multiple detections within one performance. After that, they expect a movement in the corresponding direction, which lasts long enough to be able to perform at least one step in that direction. The time constraint was adjusted according to the minimum time

it took participants to perform one step during the gesture performances, which was between 0.4 and 0.7 seconds depending on the direction.

For the actions Turn right/left there was only one gesture candidate, i.e. drawing a circle with one hand in the x-z-plane. The recognizers separated this movement into six parts, so the Turn right recognizer waited for the hand sequence movements directed right-forward, right, right-backward, backward, left-backward, and left. The Turn left recognizer was implemented symmetrically. As the users performed this gesture with quite different movement speeds, the recognizer was quite tolerant to different speeds. However, it required the circle to be drawn smoothly to cover all of the six required directions with a recognizable long enough period of time.



(a) (b) (c) (d)

Figure 7.9: Tracking image for the different recognition steps of the actions Speed up (a-c) and Stop movement (d)

The Stop movement action was implemented with a recognizer for one hand to be stretched to the front with open fingers. Therefore, it looked at the relation between the shoulder and the hand joint to ensure the correct hand position and in addition it applied a finger count recognizer for recognizing an open hand. A visualization of the extracted finger shapes during the gesture for the Stop movement action can be seen in Figure 7.9 (d). The recognizer for the gesture required a minimum of four recognized displayed fingers as can be seen in the XML definition depicted in Listing 7.1. This was found to be robust enough to ignore the fingers pointing to the front or forming a fist, but also to detect an open hand in which some fingers are relatively close to each other. For the timing, a minimum of one

second was required for the hand to stay in front, and, at the end of that phase, the detection for an open hand needed to be fulfilled for at least half a second. The last period is a bit shorter than the timings we extracted from the video performances. It was chosen this way, as the finger count recognizer often suffers from noise in the depth or tracking data and we got a more reliable recognition in this way.

Listing 7.1: XML definition of the recognizer for the stop movement gesture

```
<JointRelationRecognizer name="rightHandFront">
  <Joints main="rightHand" relative="rightShoulder"/>
  <MaxValues z="-200" />
  <MinValues y="-200"/>
</JointRelationRecognizer>
<JointRelationRecognizer name="rightHandCloseToSensor">
  <Joints main="rightHand"/>
  <MaxValues z="1750"/>
</JointRelationRecognizer>
<FingerCountRecognizer name="OpenHand">
  <Joint name="rightHand"/>
  <FingerCount min="4"/>
</FingerCountRecognizer>
<CombinationRecognizer name="RightHandFrontOpen">
  <State minDuration="0.5" >
    <!-- ignoreOnTrackingError used, so it still works if shoulder is lost -->
    <Recognizer name="rightHandFront" ignoreOnTrackingError="true"/>
    <Recognizer name="rightHandCloseToSensor"/>
  </State>
  <State minDuration="0.5">
    <Recognizer name="OpenHand"/>
    <Recognizer name="rightHandFront" ignoreOnTrackingError="true"/>
    <Recognizer name="rightHandCloseToSensor"/>
  </State>
</CombinationRecognizer>
```

The recognizers for the actions Speed up, Slow down, and Stand up were all implemented in a similar way. They all required one or both hand(s) in a specific starting position. Then they waited for a movement in backward (Speed up), downward (Slow down), or upward (Stand up) direction. After that they expected a specific end position of the hand(s). For example, the action Speed up required a starting position of both hands at least one shoulder width in front of the body (Figure 7.9 a). After a short movement in backward direction (Figure 7.9 b), the hands should be closer than one

hip width to the body (Figure 7.9 c). The timings were chosen in a way that the whole gestures took at least 0.5 to 0.7 seconds as a lot of study participants performed the movement quite fast.

7.3.2 Evaluation of the Interaction

We conducted a study to evaluate the accuracy performances of the proposed recognition system with the eleven gestural commands implemented as described in the previous section. We recruited 22 participants (7 female and 15 male) with an average age of 26 (SD=4.7) and they were all right handed. As we were testing for the accuracy of classifying predefined gestures in the system, the participants had to practice and learn each gesture, thus, their background did not matter when selected for participation.

The experiment was arranged in a room about 3 meters wide and 6.5 meters in depth. It was equipped with a 50 inch plasma display and a Microsoft Kinect for Xbox 360 placed in the center, directly below the display. When the participants entered the room, they were instructed by the administrator to stand at a point about 2 meters away from the Kinect and a description of the study was given to them. All tests were performed from the 2 meters mark, except for the gesture of the Stop movement action, where the user was asked to stand about 1.5 meters away from the screen. The participants were asked to watch a video demonstrating one of the body gestures of the eleven actions. The participants practiced the gesture and they were allowed to watch the video as many times as they wanted. When the participants said that they were ready and understood the gesture they had watched, the administrator instructed them to perform that gesture. After the first performance, the participants were asked to return to their starting point (if necessary), and repeat the gesture one more time. Then, the participants continued with the next gesture in the same way until the gestures for all actions have been performed twice. When completing the experiment, the participant is asked to fill out a short questionnaire

The eleven actions were presented to each participant in a weak counterbalanced order (Latin Square order) and the data was collected by the system automatically. Using the recognition system, described in Section

5.4, we performed the evaluation of the system for each participant. The system ran in real-time on a standard notebook computer with an Intel i7 dual core processor and 4 GB RAM. In total, 484 body gestures were analyzed by our system and the overall results are summarized in Table 7.1:

Table 7.1: Accuracy measures for the recognition of the eleven implemented gesture candidates for controlling humanoid robots

Action	**Gesture** cf. Fig. 4.3	**TP**	**FP**	**FN**	**Precision**	**Accuracy**	**Recall**
Move..							
..forward	NT (b)	42	2	0	95%	95%	100%
..backward	NT	43	2	0	96%	96%	100%
..right	NT (a)	39	0	5	100%	89%	89%
..left	NT (b)	42	1	1	98%	95%	98%
Turn..							
..left	T/NT	20	4	21	83%	44%	49%
..right	T/NT	18	3	23	86%	41%	44%
Stop	NT	32	5	7	86%	73%	82%
Speed up	NT	5	21	16	19%	12%	24%
Slow down	NT (b)	43	0	2	100%	96%	96%
Stand up	T/NT	39	1	3	98%	91%	93%
Sit down	NT (b) modified	37	5	2	88%	84%	95%
Overall		360	44	80	89%	74%	82%
..w/o Speed up		355	25	64	93%	80%	85%
..w/o Speed up, Turn left/right		317	18	20	95%	89%	94%

The results show that the overall recognition accuracy of the system for all actions was 74%, with the highest recognition rate of the Slow down action (96%) and the lowest recognition rate of the Speed up action (12%). A similar result can be seen for the recall of our system.

7.3.3 Conclusion and Discussion

The problem with the Speed up action was that we accidentally used the recognizer implemented for recognition with tracking data by PrimeSense

NiTE instead of the Microsoft Kinect SDK tracking. The gesture for Slow Down was defined by both arms stretched to the front and then coming close to the body as visualized in Figure 7.9, first three images from the left. For recognizing that the hands are close, we compared the distance along the z-axis between the hands and the torso joint (called spine in the Microsoft Kinect SDK). The maximum for that distance was defined as the hip width, which is unfortunately much smaller for the Microsoft Kinect SDK (see Figure 7.9) compared to PrimeSense NiTE. In addition, the torso joint in the Microsoft Kinect SDK is positioned a bit more backwards. Finally, when the hands are close to the body, the Microsoft Kinect SDK usually reports a reduced tracking confidence, but we requested a higher confidence in our recognizer as it would be reported by the PrimeSense NiTE tracking. Therefore, the recognizer failed in most times to detect that the hands are close to the body. The recognition for the Turn Left/Right actions also showed a much lower accuracy than the rest of the actions. In this case, it seems that the recognizers were defined in a bit too strict way, which caused the high number of false negatives. The average accuracy of the system improves to 80% by omitting the Speed up action, and to 89% by additionally omitting the actions Turn right and left.

Regarding the remaining actions, Stop movement was recognized with a slightly lower accuracy (73%) than the others. This was due to the fact that it was the only action including finger count recognizers that are in general less accurate then the other recognizers as they suffer more heavily from the distortions in the depth stream. This caused a slightly higher number of false negatives for this action.

The precision of our system was 89% on average (93% without Speed up) and herein, also the gestures for the actions Turn right and left get an acceptable number. Therefore, if our system detected a gesture performance, it usually detected the correct one. Only the Speed up action provides a low precision for the same reasons a mentioned above, all other actions have at least a precision of 83%, with some of them even reaching 100%. This achieved accuracy clearly shows encouraging results and can lead to effectively using the system in navigating a humanoid robot.

7.4 Evaluation of Gestures and Enhancements for Traveller

In this section, I will describe the implementation of the user-defined gesture set for the Traveller application described in Section 4.3. I will present the definition of the gesture recognizers, as well as how we integrated the gestural input into the interactive storytelling scenario and applied affordances, feedback and feedforward mechanisms. Finally, I will present an evaluation of the implementations.

7.4.1 Implementation of Gesture Candidates

For implementing the recognizers for the gesture candidates as combination recognizers, one first has to determine by which sequence of basic recognizers a gesture candidate can be described and how the recognizers' parameters need to be adjusted. This is done by studying sample videos of the gesture in more detail and approximating the parameters. For determining the time constraints of a combination recognizer, the measured timings as depicted in Table 4.6 can be used. However, it should again be mentioned that he timings can usually not be used directly, but should rather serve as a basis for understanding a gesture and its temporal variance between users.

Listing 7.2: XML definition of the recognizer for a head nod

```
<AngularMovementRecognizer name="HeadDown">
  <Joint name="head"/>
  <BasicAngularVelocity type="pitchDown" min="15"/>
</AngularMovementRecognizer>
<AngularMovementRecognizer name="HeadUp">
  <Joint name="head"/>
  <BasicAngularVelocity type="pitchUp" min="15"/>
</AngularMovementRecognizer>
<CombinationRecognizer name="HeadNod">
  <State maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadUp"/>
  </State>
  <State maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadDown"/>
  </State>
  <State maxDuration="0.8" timeForTransition="0.5">
```

```
17      <Recognizer name="HeadUp"/>
18    </State>
19    <State>
20      <Recognizer name="HeadDown"/>
21    </State>
22  </CombinationRecognizer>
```

To implement the gesture candidate head nod for the action yes, we used the XML definition as shown in Listing 7.2. Lines 2–5 and 6–9 define two angular movement recognizers that observe the head joint waiting for down and up pitches with at least 15 degrees per second. Lines 10–23 define the actual head nod as a combination recognizer with four states that refer to the previously defined angular movement recognizers in the sequence pitch up, down, up, down. It further restricts each state to last for no longer than 800 milliseconds and allows a pause of at most 500 milliseconds between two subsequent states. This recognizer only accepts multiple nods, but no single ones, which was done to avoid false recognitions. Nevertheless, we give hints to the user on how to perform the gesture, which is described in the next section.

The rest of the gestures were implemented in a similar way. We used combinations of joint orientation recognizers for the gestures, sit down, and turn away to check the orientations of the joints included in the gesture. For the gestures arms out we used a recognizer that combines two joint relation recognizers in one state. The first joint relation recognizer observed the left hand and shoulder and waited for the hand to be in a height similar to the shoulder (difference on the y-axis smaller than 30 centimeters) and that the hand was at least 35 centimeters left of the shoulder (using the x coordinate). The second recognizer looked for the same properties with the right hand and shoulder. The gesture candidate point to front as well used a joint relation recognizer for checking the hand position. In addition it ensured that the right elbow was not too far away from the line from the shoulder to the hand (at max 12 centimeters) and that the hand was not moving with more than 0.5 meters per second. The last condition was implemented using a linear movement recognizer. The candidate pointing at one after the other was defined in the same way but with an additional state that allowed hand movement in between the two pointing states. We also

implemented the second gesture candidate for the action ask permission that was the gesture tip on shoulder. This was basically realized the same way as point to front, but in addition, it waited for a sequence of linear movements in upward and downward direction of the hand. The gesture candidates step forward and step forward were implemented as linear movement recognizers looking at the torso joint and waiting for a movement in z or −z direction after a short duration of standing still.

Although we implemented the recognizers for all gesture candidates, we had to decide which of the candidates we would actually use in the three cases in which two candidates were determined. As they seemed to fit a bit better to the parallel and surrounding gestures and because of their partly more reliable recognizers, we chose the gesture turn away for the action leave bar, point to front for ask guard to talk to supervisor, and tip on shoulder for ask permission. The recorded videos of the gesture performances were very useful to create the XML definition for the gestures as well as the measured timings for the gestures' stroke phases. Nevertheless, the creation of a gesture recognizer based on this data is still a challenging task that has to be done in a careful way to realize the gestures at least close to as they were intended by the users. In the further development process, we iteratively added more gestures to the gesture set which finally lead to 24 different gestures developed for 50+ in-game actions in Traveller. The additional gestures were needed to include more content at a later stage, but also to include gestures that are uncommon for the participants, e.g. the bow as a formal greeting gesture.

Affordances, Feedback, and Feedforward

We applied affordances, feedforward and feedback in the Traveller application by integrating the gestures with onscreen symbols, and adding the live tracking image of the user. To assist the users in performing the gestures, we displayed labels for the corresponding actions and paired them with symbols that visualized how the gestures for these actions should be performed. Static images represented postures, animated image sequences and arrows in the image visualized motions. For fitting to the comic-like graphics style

of Traveller and as they could easily be generated out of the recording of a person's gesture recording, we decided to use cyan-colored tracking shapes and skeletons for visualizing the gestures. All gesture symbol were linked with a recognizer defined in FUBI's gesture XML and an ingame event they should trigger. As long as a symbol was displayed on the screen, the recognition framework checked the linked recognizer for currently tracked users. In case a recognition was successful, the corresponding ingame event was triggered. Figure 7.10 depicts a scene of the first CI displaying three symbols used to help generate the gestures.

Figure 7.10: Three symbols displaying interaction options in Traveller

In the lower right corner of the screen (see Figure 7.10), users could further see their tracked body shape and skeleton to compare it against the gesture symbol. This was especially emphasized by the fact that the tracking image looked similar to the gesture symbol.

7.4.2 Enhancement with a Swipe Menu

In our gesture study, we already had multiple actions that received identical gesture candidates and – in parallel – had relatively low agreement scores. This was further emphasized in the later development of Traveller. For still being able to distinguish between those actions, we additionally integrated a graphical menu that was controlled by freehand interaction and included

further dialogue options as shown in Figure 7.11. The interface still used full body interaction, but allowed for any arbitrary in-game action to be included. Integrating two different input types had to be done carefully to provide good usability. Therefore, we completely separated the gesture and swipe interaction. Whenever there were too complex dialogue options available, we displayed a special gesture (see Figure 7.11 in the left-hand image) which opened a menu that included the additional dialogue options as shown in Figure 7.11 on the right-hand side. Apart from the available con-

Figure 7.11: Traveller dialogue menu – *Left image:* Gesture for opening the dialogue menu *Right image:* Dialogue menu controlled by swiping

versational actions the menu also displayed one option cancel the dialogue selection. Within the menu, the options were arranged around a circle in the middle of the screen, with each of them occupying an equally sized sector around the middle circle. For selecting one of the options in the dialogue menu, users first had to stretch out their hand to the front, wait until the menu gets activated, and then perform a swiping gesture in the direction of the option they would like to select. Activation of the menu was visualized by the middle circle changing its color from blue to yellow. In addition, the circle always contains textual instructions for what to do next. As soon as the start of a swiping gesture was recognized, the corresponding arrow got a little stretched in its pointing direction, and also changed its color to yellow together with the background of the corresponding action text. As soon as the swipe was completed and thus a selection was performed

successfully, a sound was played for additional feedback. The swipe interaction approach enabled us to have an interesting story with as many and as complex actions as we wanted without worrying about how all of them could be represented by unambiguous gestures. However, the interaction modality remained the same for all in-game actions, and the two interaction types were similar enough to provide a fluent user experience. In the final Traveller version, 17 different interaction options were integrated using the dialogue menu approach. In a single walk through the complete story, an average user selected about 50 different actions (40 by gestures and 10 by swipes) and further had to perform 36 swipes for selecting "continue".

7.4.3 Evaluation of the Interaction

To evaluate our interaction approach, we conducted a user study with two groups of participants: one interacting with the actual full body interaction interface, and the other one interacting with a traditional mouse interface. In the latter case, we only displayed the action labels, but omitted the gesture symbols (see Figure 7.12). The participants controlled a mouse cursor and selected in-game actions by moving the cursor over the label and pressing the left mouse button.

Figure 7.12: Hidden gesture symbols during mouse interaction in Traveller

With this study setup, we were not only able to test the robustness of our gesture recognition, but we could also compare its usability and user

experience to a traditional interaction system. Although we had carefully developed the full body interaction, we expected that the well-known mouse interface would still provide better usability in parallel to the findings of Dow et al. [41]. Nevertheless, our goal regarding recognition accuracy was to stay above 90% to prove a robust interaction system. Regarding user experience, we expected to find advantages for our more human-oriented interaction approach, especially regarding immersion (in line with Dow et al. [41]). However, it was uncertain whether it improved the users' engagement in the interaction, which would be in line with Aylett et al. [6] who investigated handheld devices, but could not be found by Dow et al. [41] regarding speech and embodied interaction.

Setup and Procedure

The experiment was set up in a room of about 3 meters width and 6.5 meters depth. The participants were standing at a distance of about 2.5 meters in front of a 50 inch display which had a Microsoft Kinect for Xbox 360 placed centred below it. The participants were told to return to the initial position whenever a gesture requires them to temporarily move away. An experimenter was sitting to the left of the participants to take notes about the gestures that participants performed and how they were recognized by our system. The participants should follow a "think aloud protocol", which meant that they spoke out the action they wanted to choose before actually performing the corresponding gesture to trigger it. In this way, the experimenter knew whether the system recognized the intended gesture or not. For the mouse condition, the participants were sitting in front of a 26 inch screen controlling the application with a standard mouse. The experimenter was still sitting left of them, however, there was no need for him to take notes about the recognition of inputs in this case.

After a short introduction and a demographic questionnaire, the experimenter explained the participants their role in the study and gave an introduction on the study procedure. Then, the participants could try out the interaction during the initial scene in the Grandmother's café, before the actual evaluation part started that covered the bar scene and the two

subsequent museum scenes (= the country Malahide). In this way, the evaluation covered more than half of the interaction, but usually didn't last for more than half an hour per participant. Directly after the second museum scene, the participants filled in a questionnaire that contained six question about usability of the system derived from the System Usability Scale (SUS) [20], namely "easy to use", "easy to learn", "intuitive to use" and negatively formulated counterparts. The participants further filled in the "in-game" version of the Game Experience Questionnaire [75] with two questions for each of the components competence (how good did the participants think they performed), sensory and imaginative immersion (how much were they immersed in Traveller's presentation), flow (did they fully concentrate on the game and forget about the outside world), tension (were they anxious about the events in Traveller), challenge (did they feel challenged), negative affect (were they bored) and positive affect (did they enjoy the game). After that, we included four questions about spatial presence that basically asked the participants how much they felt themselves as a part of the virtual environment and able to act within it. At the end, we further asked 13 questions about how the participants perceived the characters' behaviors and how they reacted to the user inputs. All question were answered on a five-point Likert scale ranging from 1 (not agree at all) to 5 (fully agree).

Results and Discussion

Note that the results have been updated in comparison to the ones published in [96] as we included more participants. The group that used full body interaction consisted of 15 participants (2 females, average age 24.9), all with German cultural background and right-handed. On a scale from 1 (no experience) to 5 (almost daily usage), they stated themselves a lower medium experience with body gesture based interaction, minimum 1, maximum 5, median 3. The participants also indicated good knowledge of the English language on a scale from 1 (basic knowledge) to 5 (native speaker), minimum 2, maximum 4, median 3 (=fluently spoken and written). Overall they performed 360 gestures and 312 swipes. The experimenter counted gestures correctly recognized by the system (true positives = TP), gestures

yielded by the system while participants performed a different gesture or nothing at all (false positives = FP), gestures performed by the participants, but nothing recognized by the system (false negatives = FN), and gestures obviously wrongly performed by the participants (user errors = UE), e.g. when the gesture is performed with the wrong hand or in the opposite direction. For both gesture and swipe interaction, we calculated precision (= $\frac{TP}{TP+FP}$), accuracy (= $\frac{TP}{TP+FP+FN}$), recall (= $\frac{TP}{TP+FN}$) and user error rate (= $\frac{UE}{TP+FP+FN+UE}$). Our results are shown in Table 7.2:

Table 7.2: Accuracy measures for the recognition of the full body gestures and freehand GUI selections in Traveller

	TP	**FP**	**FN**	**UE**	**Precision**	**Accuracy**	**Recall**	**UE-Rate**
Gestures	335	1	15	9	99,70%	95,44%	95,71%	2,50%
Swipes	312	24	37	16	92,86%	83,65%	89,40%	4,11%
Overall	647	25	52	25	96,28%	89,36%	92,56%	3,34%

Both our full body interaction types received high recognition rates, with even better results for the gestures. A Pearson's chi-square test confirmed significantly less recognition errors ($FP + FN$) for the gestures in comparison to the swipes, $\chi^2(1) = 26.47$, $p < 0.001$. This states that the gesture interaction, which we consider to be the more intuitive type of interaction, was also better recognized better by the system. Some of the false positive swipes were caused by an error in the program code that in some cases selected a dialogue option without user interaction. The issue was fixed in the meantime and we assume that the accuracy for the swipe interaction would now be closer to its recall value. However, even without taking into account any false positives, the significant difference between gesture and swipe interaction is still present in the current data. Overall, this confirms that a depth sensor enables applying robust real-time recognition of body gestures without handheld devices. In addition, we introduce a swipe menu that can be used to enhance the body gestures and to allow for more complex interactions within the same modality.

In the group with mouse interaction, we had 10 participants (2 females, average age 24.6), all with German cultural background, one left-handed.

Again, they indicated good knowledge of the English language, minimum 2, maximum 4, median 3. We measured the time it took the participants to go through the evaluated part of the story (first action selection in the beach bar to last action selection in the second museum scene) and the number of actions they selected during that period. With mouse interaction, they needed 5:50–7:42 minutes (M = 6:30 minutes, SD = 0:37 minutes) and selected 25–39 actions (M = 31.4, SD = 3.69), while they spent significantly more time with the gesture-based interaction ($t(14.60) = 2.44$, $p < 0.05$, $r = 0.54$) with 5:06–22:28 minutes (M = 9:47 minutes, SD = 5:06 minutes) and they selected 26–40 actions (M = 32.8, SD = 4.38). Although the average duration with gesture interaction was significantly longer than with mouse interaction, an interesting finding here is that the fastest users actually came out of the gesture group. The final questionnaires revealed a higher usability ($t(23) = 2.55$, $p < 0.05$, $r = 0.47$) for the mouse in comparison to the Kinect, however, all other parts did not show any significant differences.

7.4.4 Conclusion and Future Work

In Traveller, we implemented two different full body interaction types: in one, users had to perform full body gestures as visualized by on-screen symbols, in the other one, users controlled a circular menu with freehand swiping gestures. In a user study, the two full body interaction types were evaluated and compared to a traditional mouse interface. Both our full body interaction types received good recognition rates. However, only the gesture interaction reached our goal of 90% recognition accuracy, while the swipe interaction stayed slightly below it. In comparison with the mouse interface, participants spend significantly more time in the game when using full body interaction. However, the novel interaction did not necessarily slow down the participants, as the fastest participants actually were using this interaction modality. Furthermore, our results indicate higher usability with the mouse, but there was no difference for the two modalities regarding game experience or the perception of the characters.

Although we also did not receive significant differences regarding immersion or spatial presence, the full body interaction might get higher scores

when using it in a virtual reality setup. In addition, the application did not yet implement all options to support the user during the interaction such as automatic feedback why a gesture has not been recognized, and step-wise instructions for more complex gestures as described in Section 6.1. With those helping mechanisms included, Traveller might receive better usability and game experience ratings.

Chapter 8

Conclusions

Full body interaction was introduced to the mass market thanks to the introduction of the Kinect in 2010. It allows users to interact without handheld devices, but through motions of the whole body. While this novel type of interaction offers a lot of new possibilities, it also brings new challenges for designers, developers, and end users, as it was described in Section 1.4. Therefore, the overall goal of this dissertation was to find improvements for the process of creating full body interaction applications, while tackling the challenges mentioned in the introduction. This was done according to five steps, which match the work presented in Chapters 3–7. In this chapter, I will summarize my contributions and finish this dissertation with directions for future research.

8.1 Contributions

In an **exploration of full body interaction**, I first compared full body gestures with freehand GUI interaction. I found that – especially for virtual environments – it can be worth the effort to implement more complex but also more natural body gestures, instead of trying to port the conventional point-and-click paradigm to full body interaction. I looked at four main interaction tasks in a virtual environment and compared which of the

modalities gestures and speech users would choose for different tasks. While gestural interaction was clearly chosen for navigational tasks, speech was still mainly chosen for dialogues. However, the selection and manipulation tasks showed no clear preferences.

I then investigated interaction design with **user-defined gestures**. In a first task, I tried to categorize types of full body interaction in general and full body gestures in particular. Full body interaction was split into **continuous and discrete interaction** and I developed a **taxonomy of full body gestures**. As existing taxonomies did not target full body interaction or were only focused on specific application scenarios, there was a need for a new taxonomy. The taxonomy for full body gestures was based on fundamental research of the social sciences. While applying the taxonomy, user-defined gesture sets were created in order to provide intuitive gesture sets for two application scenarios. For this purpose, I adopted and modified the process of Wobbrock et al. [189], as previous work on user-defined gestures focused on other interaction modalities and application scenarios, which have important differences to full body interaction.

The main technical contribution of the dissertation is the freely-available and open-source **FUBI framework**, which helps developers to prototype, test and iteratively implement full body interaction in their system. While FUBI therefore includes several traditional gesture recognition algorithms, it also provides new **gesture recognition techniques**, which were necessary as the recognition of gestures within unsegmented three-dimensional multi-point data poses additional challenges that were often omitted in the past. The basis of FUBI's gesture recognition system is its XML-based **full body gesture definition language** which also provides a **coding scheme of full body gestures**. Gestures can easily be defined directly in XML or they can be created using the generator tool in the FUBI GUI. Nevertheless, the system can recognize a wide range of gestures through its support of simple rule-based gesture definitions as well as more complex gestures defined by finite state machines or sample recordings in the case of template recognizers. In comparison to existing coding schemes, the scheme based on FUBI's gesture definition language is directly entwined with the

gesture recognition technique and supports many options to accurately define certain gestures.

The FUBI framework further provides multiple techniques for providing **affordances, feedforward and feedback mechanisms** to **support users during the interaction**. This area of research is still relatively new and many research directions are still open. Nevertheless, in Chapter 6, I investigated different supporting techniques and found preferences for certain gestures visualizations. The findings can help designers of full body interaction to decide, which techniques they want to use in their applications and to know, what they have to take care of.

In a last step, the FUBI framework with its gesture recognition capabilities as well as its affordances and feedback/-forward mechanisms was used to implement multiple **sample applications**, incorporating both continuous interaction with an **avatar control** and multiple **freehand GUI techniques**, as well as discrete interaction with actual **full body gestures**. I reported the results of multiple **evaluation studies**, which additionally validated the work presented in the preceding steps.

8.2 Future Work

This dissertation provided a first step to ease the process of creating full body interaction. In its main contributions, I presented a method for creating intuitive gesture sets to ease the interaction design. I further created the FUBI framework, so that developers can implement full body interaction for their application more easily, even without a great background in gesture recognition. I finally investigated methods for supporting users during full body interaction.

I believe that full body interaction will get more and more important in the future and therefore, more research is needed. There are many potential directions, in which the work of this dissertation could be continued. I will describe some of those directions in the following sections.

8.2.1 Enhanced Design Techniques

In Chapter 4, I comprehensively investigated a technique for including the user in the design process of different applications employing full body interaction. In this way, the interaction designer can create a user-defined gesture set, which is considered more intuitive than a manually defined one. However, other important aspects of the interface design might be handled better by using formal mechanisms, e.g. similar to Fitts's Law [49]. Sometimes it might as well be a good choice to use an iterative method, e.g similar to the RITE method [123], to improve the interface in multiple steps. Both options could improve the usability of the gesture set and also help at implementing the actual recognizers of the set.

8.2.2 Enhanced Tracking

FUBI supports different tracking sensors and hardware and further incorporates a simple finger recognition technique to enhance the tracking software that only reports hand transformations. It also supports the LEAP Motion Controller to provide more accurate finger tracking independent from the default user tracking, although the collaboration is not optimal. The LEAP Motion Controller targets a scenario, in which the user is sitting in front of a screen, but this is in conflict with the default full body interaction scenario, in which the user is standing about two meters away from a screen. Especially when the sensing hardware improves, it will become necessary to support a more detailed hand model and finger poses. Future versions of the tracking software might already include such a hand model that FUBI could use, else it might be necessary to develop an own one in parallel with a more advanced finger recognition technique.

The Kinect SDK already includes facial tracking features. Although they are not very precise at the moment, FUBI could support to recognize facial expressions in a future version.

8.2.3 Enhanced Gesture Recognition

FUBI already covers a wide range of gesture recognition technologies, however, it does not use techniques that usually require a larger amount of training data such as HMMs or statistical classifiers. Nevertheless, for actual commercial systems, such recognition technologies might be a better choice as they could support a higher robustness for that effort. FUBI's TemplateRecognizers, which are closest to such techniques, use a computationally very expensive temporal data segmentation technique by either using a window of fixed length or looking for an optimal window length using a golden section search strategy. There is no general solution to achieve a faster and more efficient segmentation at the moment, but more research could be done in this direction.

Other additional recognition technologies could use a scripting language for defining gestures similar to XDKinect [130] or more options for concatenating basic recognizers in the combinations similar as in GestIT [165]. However, those techniques would make the gesture definition more complicated and it is unclear if this higher level of complexity is really needed.

Apart from those techniques, the combination recognizers could also use variables that transport information across the borders of the FSM's different states. For example, when looking at the gesture "forming a chest" of Figure 5.18c, this could be used to ensure that all sides of the "chest" have a similar length. In addition, the states could not completely abort the recognition process after a certain interruption, but only lower a confidence value until it reaches zero. In this way, the FSM-based combinations would be less strict and provide a more "fuzzy" output similar to HMMs.

FUBI already includes sample definition files that realize parts of the Expo coding system [172]. Apart from that, the gesture definition language could directly support terms of existing coding schemes such as the BAP system [33] or at least provide a converter for translating other gesture definitions into the FUBI language to make them usable there as well.

8.2.4 Enhanced Helping Mechanisms

Regarding the techniques for supporting users during full body interaction, there are still many options open to continue the work of this dissertation. For example, in Section 6.1.2, a first implementation of gesture symbols automatically generated out of FUBI's gesture XML was presented. It also incorporated a first version of automatically generated affordances and feedback/-forward information. However, this section did not cover all ways to visualize this information, e.g. it did not investigate schematic drawings, stereoscopic rendering, multi-angle view, etc. A study to investigate more options similar to the one described in Section 6.2 could provide more insights on the advantages and disadvantages of visualization techniques.

8.2.5 Additional Application Scenarios

As mentioned in Section 2.7.3, full body interaction is a promising option for virtual and augmented reality applications as it supports a close-to-real-life interaction. We have already started to investigate the potential of this combination in our work [34], in which we used a motion capturing suit to allow for seamless interaction with virtual content that augments the real environment. The results were promising, however, the interaction was still very limited and the scenario itself was not elaborated enough to get strong findings. Instead of using a motion capturing suit, it might also be possible to use multiple Kinects to allow for interaction independent of the overall orientation of the user.

Other application scenarios not covered in this dissertation are collaborative scenarios which have been investigated briefly in Section 3.1 and scenarios in which it is undesired or impossible to touch anything such as when using an endoscope as in the example of Section 1.3.

8.2.6 Multimodal Interaction

This dissertation clearly focused on the single modality of full body interaction, however, a first hint on its possible combinations with other modalities was given in Section 3.2. Because both modalities are close to real-life in-

teraction, it makes sense to combine gestural interaction with speech to join their strengths and to achieve a more immersive experience. This was also intended with the Kinect, as both, the first and second generation, include a microphone array for speech input. Depending on the application scenario, other modalities can be combined with full body interaction as well, e.g. interaction with tangibles or the integration of hand-held devices can be used in case the scenario profits from the integration of real-life objects.

Bibliography

[1] Abeel, T., Van de Peer, Y., and Saeys, Y. Java-ML: A machine learning library. *The Journal of Machine Learning Research*, 10:931–934, 2009. [cited at p. 48]

[2] Alon, J., Athitsos, V., Yuan, Q., and Sclaroff, S. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1685–1699, 2009. [cited at p. 33, 41]

[3] Anderson, D., Bailey, C., and Skubic, M. Hidden markov model symbol recognition for sketch-based interfaces. In *Proc. AAAI Fall Symposium 2004*, pages 15–21, 2004. [cited at p. 32, 33]

[4] Anderson, F., Grossman, T., Matejka, J., and Fitzmaurice, G. YouMove: Enhancing movement training with an augmented reality mirror. In *Proc. UIST 2013*, pages 311–320. ACM, 2013. [cited at p. 61, 167, 168]

[5] Anthony, L. and Wobbrock, J. O. $N-protractor: A fast and accurate multistroke recognizer. In *Proc. GI 2012*, pages 117–120. Canadian Information Processing Society, 2012. [cited at p. 35]

[6] Aylett, R., Vannini, N., André, E., Paiva, A., Enz, S., and Hall, L. But that was in another country: Agents and intercultural empathy. In *Proc. AAMAS 2009*, pages 329–336. IFAAMAS, 2009. [cited at p. 68, 105, 210]

[7] Bailenson, J. N., Blascovich, J., Beall, A. C., and Loomis, J. M. Equilibrium theory revisited: Mutual gaze and personal space in virtual environ-

ments. *Presence: Teleoperators and Virtual Environments*, 10(6):538–598, 2001. [cited at p. 179, 183]

[8] Bailly, G., Walter, R., Müller, J., Ning, T., and Lecolinet, E. Comparing free hand menu techniques for distant displays using linear, marking and finger-count menus. In *Proc. INTERACT 2011*, pages 248–262. Springer Berlin Heidelberg, 2011. [cited at p. 148]

[9] Barattini, P., Morand, C., and Robertson, N. M. A proposed gesture set for the control of industrial collaborative robots. In *Proc. RO-MAN 2012*, pages 132–137, 2012. [cited at p. 94]

[10] Bau, O. and Mackay, W. E. Octopocus: a dynamic guide for learning gesture-based command sets. In *Proc. UIST 2008*, pages 37–46. ACM, 2008. [cited at p. 60, 61]

[11] Baytaş, M. A., Yemez, Y., and Özcan, O. Hotspotizer: End-user authoring of mid-air gestural interactions. In *Proc. NordiCHI 2014*, pages 677–686. ACM, 2014. [cited at p. 47, 50]

[12] Beder, C., Bartczak, B., and Koch, R. A comparison of PMD-cameras and stereo-vision for the task of surface reconstruction using patchlets. In *Proc. CVPR 2007*, pages 1–8. IEEE, 2007. [cited at p. 20]

[13] Bhuyan, M., Ghosh, D., and Bora, P. Continuous hand gesture segmentation and co-articulation detection. In Kalra, P. and Peleg, S., editors, *Computer Vision, Graphics and Image Processing*, volume 4338 of *Lecture Notes in Computer Science*, pages 564–575. Springer Berlin Heidelberg, 2006. [cited at p. 41]

[14] Blanchard, C., Burgess, S., Harvill, Y., Lanier, J., Lasko, A., Oberman, M., and Teitel, M. Reality built for two: A virtual reality tool. In *Proc. I3D 1990*, pages 35–36. ACM, 1990. [cited at p. 69]

[15] Bleiweiss, A., Eshar, D., Kutliroff, G., Lerner, A., Oshrat, Y., and Yanai, Y. Enhanced interactive gaming by blending full-body tracking and gesture animation. In *Proc. SA 2010*. ACM, 2010. [cited at p. 60, 133, 134]

[16] Bobick, A. F. and Wilson, A. D. A state-based approach to the representation and recognition of gesture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1325–1337, 1997. [cited at p. 29, 38]

[17] Bodiroža, S., Stern, H. I., and Edan, Y. Dynamic gesture vocabulary design for intuitive human-robot dialog. In *Proc. HRI 2012*, pages 111–112. ACM, 2012. [cited at p. 94]

[18] Bolt, R. A. "Put-that-there": Voice and gesture at the graphics interface. In *Proc. SIGGRAPH 1980*, pages 262–270. ACM, 1980. [cited at p. 16]

[19] Broccia, G., Livesu, M., and Scateni, R. Gestural interaction for robot motion control. In *Proc. EuroGraphics Italian Chapter 2011*, pages 61–66. The Eurographics Association, 2011. [cited at p. 70]

[20] Brooke, J. SUS - a quick and dirty usability scale. *Usability Evaluation in Industry*, 189:194, 1996. [cited at p. 186, 211]

[21] Bull, P. E. *Posture and Gesture*. International series in experimental social psychology. Pergamon Press, 1987. [cited at p. 25]

[22] Cabibihan, J., So, W.-C., and Pramanik, S. Human-recognizable robotic gestures. *IEEE Transactions on Autonomous Mental Development*, 4(4):305–314, 2012. [cited at p. 70]

[23] Calinon, S., Guenter, F., and Billard, A. On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007. [cited at p. 38, 126, 154]

[24] Casiez, G., Roussel, N., and Vogel, D. 1 € filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proc. CHI 2012*, pages 2527–2530. ACM, 2012. [cited at p. 125, 126, 138]

[25] Cavazza, M., Charles, F., Mead, S., Martin, O., Marichal, X., and Nandi, A. Multimodal acting in mixed reality interactive storytelling. *IEEE Multimedia*, 11(3):30–39, 2004. [cited at p. 67, 83, 86]

[26] Cavazza, M., Charles, F., and Mead, S. J. Interacting with virtual characters in interactive storytelling. In *Proc. AAMAS 2002*. ACM, 2002. [cited at p. 67]

[27] Cavazza, M., Lugrin, J.-L., Pizzi, D., and Charles, F. Madame bovary on the holodeck: Immersive interactive storytelling. In *Proc. MULTIMEDIA 2007*, pages 651–660. ACM, 2007. [cited at p. 67]

[28] Chen, T., Seidel, H.-P., and Lensch, H. Modulated phase-shifting for 3D scanning. In *Proc. CVPR 2008*, pages 1–8. IEEE, 2008. [cited at p. 21]

[29] Cohen, P., McGee, D., Oviatt, S., Wu, L., Clow, J., King, R., Julier, S., and Rosenblum, L. Multimodal interaction for 2D and 3D environments. *IEEE Computer Graphics and Applications*, 19(4):10–13, 1999. [cited at p. 67]

[30] Colombo, C., Del Bimbo, A., and Valli, A. A real-time full body tracking and humanoid animation system. *Parallel Computing*, 34(12):718–726, 2008. [cited at p. 22]

[31] Corradini, A. Dynamic time warping for off-line recognition of a small gesture vocabulary. In *Proc. Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems Workshop on ICCV 2001*, pages 82–89, 2001. [cited at p. 33]

[32] Corradini, A. and Cohen, P. On the relationships among speech, gestures, and object manipulation in virtual environments: Initial evidence. In *Proc. Workshop on Natural, Intelligent and Effective Interaction in Multimodal Dialogue Systems 2002*, 2002. [cited at p. 67, 83, 86]

[33] Dael, N., Mortillaro, M., and Scherer, K. The body action and posture coding system (BAP): Development and reliability. *Journal of Nonverbal Behavior*, 36(2):97–121, 2012. [cited at p. 26, 219]

[34] Damian, I., Kistler, F., Obaid, M., Büling, R., Billinghurst, M., and André, E. Motion Capturing Empowered Interaction with a Virtual Agent in an Augmented Reality Environment. In *Proc. Symposium on Mixed and Augmented Reality 2013*. IEEE, 2013. [cited at p. 220]

[35] Davis, J. and Shah, M. Visual gesture recognition. *IEE Proc. Vision, Image and Signal Processing*, 141(2):101–106, 1994. [cited at p. 29]

[36] Degens, N., Jan, G., Mascarenhas, S., Silva, A., Paiva, A., Kistler, F., André, E., Aylett, R., and Kappas, A. Traveller–interacting with agents

to deal with misunderstandings due to culture. In *Proc. FDG 2013*. ACM, SASDG Digital Library, 2013. [cited at p. 106]

[37] Degens, N., Jan, G., Mascarenhas, S., Silva, A., Paiva, A., Kistler, F., André, E., Swiderska, A., Krumhuber, E., Kappas, A., Hume, C., Hall, L., and Aylett, R. Traveller - intercultural training with intelligent agents for young adults. In *Proc. IDGEI 2013 on FDG 2013*. ACM, SASDG Digital Library, 2013. [cited at p. 106]

[38] Deng, J. and Tsui, H. An HMM-based approach for gesture segmentation and recognition. In *Proc. Pattern Recognition 2000*, volume 3, pages 679–682, 2000. [cited at p. 38]

[39] Dias, J., Mascarenhas, S., and Paiva, A. FAtiMA modular: Towards an agent architecture with a generic appraisal framework. In Bosse, T., Broekens, J., Dias, J., and van der Zwaan, J., editors, *Emotion Modeling*, volume 8750 of *Lecture Notes in Computer Science*, pages 44–56. Springer International Publishing, 2014. [cited at p. 108]

[40] Don, L. and Smith, S. P. Applying bimanual interaction principles to text input on multi-touch surfaces and tabletops. In *Proc. ITS 2010*, pages 253–254. ACM, 2010. [cited at p. 136, 137, 141]

[41] Dow, S., Mehta, M., Harmon, E., MacIntyre, B., and Mateas, M. Presence and engagement in an interactive drama. In *Proc. CHI 2007*, pages 1475–1484. ACM, 2007. [cited at p. 67, 105, 210]

[42] Dunlop, M. and Levine, J. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. In *Proc. CHI 2012*, pages 2669–2678. ACM, 2012. [cited at p. 136]

[43] Efron, D. *Gesture and Environment*. King's Crown Press, 1941. [cited at p. 23]

[44] Ekman, P. and Friesen, W. V. Hand movements. *Journal of Communication*, 22(4):353–374, 1972. [cited at p. 23, 26]

[45] Ende, T., Haddadin, S., Parusel, S., Wusthoff, T., Hassenzahl, M., and Albu-Schaffer, A. A human-centered approach to robot gesture based communication within collaborative working processes. In *Proc. IROS 2011*, pages 3367–3374. IEEE, 2011. [cited at p. 94]

[46] Endrass, B., Rehm, M., Lipi, A.-A., Nakano, Y., and André, E. Culture-related differences in aspects of behavior for virtual characters across germany and japan. In *Proc. AAMAS 2011*, pages 441–448. IFAAMAS, 2011. [cited at p. 68]

[47] Ferraro, G. P. *The cultural dimension of international business*. Prentice Hall, 1998. [cited at p. 178]

[48] Findlater, L. and Wobbrock, J. Personalized input: improving ten-finger touchscreen typing through automatic adaptation. In *Proc. CHI 2012*, pages 815–824. ACM, 2012. [cited at p. 136]

[49] Fitts, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381–391, 1954. [cited at p. 218]

[50] Freeman, D., Benko, H., Morris, M. R., and Wigdor, D. Shadowguides: Visualizations for in-situ learning of multi-touch and whole-hand gestures. In *Proc. ITS 2009*, pages 165–172. ACM, 2009. [cited at p. 60]

[51] Freund, Y. and Schapire, R. E. A desicion-theoretic generalization of on-line learning and an application to boosting. In Vitányi, P., editor, *Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin Heidelberg, 1995. [cited at p. 28, 48]

[52] Frijda, N. H. Emotion, cognitive structure, and action tendency. *Cognition and emotion*, 1(2):115–143, 1987. [cited at p. 26]

[53] Frijda, N. H., Kuipers, P., and Ter Schure, E. Relations among emotion, appraisal, and emotional action readiness. *Journal of personality and social psychology*, 57(2):212, 1989. [cited at p. 26]

[54] Fuccella, V. and Costagliola, G. Unistroke gesture recognition through polyline approximation and alignment. In *Proc. CHI 2015*, pages 3351–3354. ACM, 2015. [cited at p. 36, 126, 156]

[55] Gabbard, J., Hix, D., and Swan, J. User-centered design and evaluation of virtual environments. *IEEE Computer Graphics and Applications*, 19(6):51–59, 1999. [cited at p. 54]

[56] Gillian, N. and Paradiso, J. A. The gesture recognition toolkit. *The Journal of Machine Learning Research*, 15(1):3483–3487, 2014. [cited at p. 48]

[57] Gleeson, B., MacLean, K., Haddadi, A., Croft, E., and Alcazar, J. Gestures for industry intuitive human-robot communication from human observation. In *Proc. HRI 2013*, pages 349–356. IEEE, 2013. [cited at p. 94]

[58] Good, M. D., Whiteside, J. A., Wixon, D. R., and Jones, S. J. Building a user-derived interface. *Communnications of the ACM*, 27(10):1032–1043, 1984. [cited at p. 55]

[59] Grossman, T. and Balakrishnan, R. The bubble cursor: Enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proc. CHI 2005*, pages 281–290. ACM, 2005. [cited at p. 52, 138]

[60] Gu, Y., Do, H., Ou, Y., and Sheng, W. Human gesture recognition through a Kinect sensor. In *Proc. ROBIO 2012*, pages 1379–1384, 2012. [cited at p. 39]

[61] Guigar, B. *The Everything Cartooning Book: Create Unique And Inspired Cartoons For Fun And Profit*. Everything Books. F+W Media, 2004. [cited at p. 168]

[62] Hachaj, T. and Ogiela, M. Rule-based approach to recognizing human body poses and gestures in real time. *Multimedia Systems*, 20(1):81–99, 2014. [cited at p. 46, 50]

[63] Hall, E. T. *The Hidden Dimension*. Doubleday, 1966. [cited at p. 178]

[64] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. [cited at p. 48]

[65] Hartson, R. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & Information Technology*, 22(5):315–338, 2003. [cited at p. 58, 59]

[66] Hilliges, O., Kim, D., Izadi, S., Weiss, M., and Wilson, A. Holodesk: Direct 3D interactions with a situated see-through display. In *Proc. CHI 2012*, pages 2421–2430. ACM, 2012. [cited at p. 69]

[67] Hincapié-Ramos, J. D., Guo, X., Moghadasian, P., and Irani, P. Consumed endurance: A metric to quantify arm fatigue of mid-air interactions. In *Proc. CHI 2014*, pages 1063–1072. ACM, 2014. [cited at p. 52]

[68] Hofstede, G. J., Pedersen, P. B., and Hofstede, G. *Exploring Culture - Exercises, Stories and Synthetic Cultures*. Intercultural Press, 2002. [cited at p. 178]

[69] Hofstede, G. J. Role playing with synthetic cultures: the evasive rules of the game. *Experimental Interactive Learning in Industrial Management: New approaches to Learning, Studying and Teaching*, page 49, 2005. [cited at p. 107]

[70] Hong, P., Turk, M., and Huang, T. S. Gesture modeling and recognition using finite state machines. In *Proc. Automatic Face and Gesture Recognition 2000*, pages 410–415. IEEE, 2000. [cited at p. 29, 30]

[71] Hoste, L., Dumas, B., and Signer, B. SpeeG: a multimodal speech- and gesture-based text input solution. In *Proc. AVI 2012*, pages 156–163. ACM, 2012. [cited at p. 66, 194, 195]

[72] Hu, C., Meng, M., Liu, P., and Wang, X. Visual gesture recognition for human-machine interface of robot teleoperation. In *Proc. IROS 2003*, pages 1560–1565. IEEE, 2003. [cited at p. 70]

[73] Huckauf, A. and Urbina, M. Gazing with pEYE: new concepts in eye typing. In *Proc. APGV 2007*, pages 141–141. ACM, 2007. [cited at p. 53]

[74] Ibañez, R., Soria, Á., Teyseyre, A., and Campo, M. Easy gesture recognition for Kinect. *Advances in Engineering Software*, 76(0):171–180, 2014. [cited at p. 47, 50]

[75] IJsselsteijn, W., de Kort, Y., Poels, K., Jurgelionis, A., and Bellotti, F. Characterising and measuring user experiences in digital games. In *Methods for Evaluating Games – How to Measure Usability and User Experience in Games Workshop held on ACE 2007*, 2007. http://gamexplab.nl/includes/pages/publications/articles/IJsselsteijn et al 2007 Characterising and Measuring User Experiences ACE 2007 workshop.pdf (accessed 2015-10-13). [cited at p. 211]

[76] Ingmarsson, M., Dinka, D., and Zhai, S. TNT: a numeric keypad based text input method. In *Proc. CHI 2004*, pages 639–646. ACM, 2004. [cited at p. 136]

[77] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. In *Proc. UIST 2011*, pages 559–568. ACM, 2011. [cited at p. 69]

[78] Jan, D., Herrera, D., Martinovski, B., Novick, D., and Traum, D. A computational model of culture-specific conversational behavior. In *Proc. IVA 2007*, pages 45–56. Springer Berlin Heidelberg, 2007. [cited at p. 68]

[79] Janowski, K., Kistler, F., and André, E. Gestures or speech? Comparing modality selection for different interaction tasks in a virtual environment. In *Proc. Tilburg Gesture Research Meeting 2013*. http://tiger.uvt.nl/pdf/papers/janowski.pdf (accessed 2015-9-15), 2013. [cited at p. 80]

[80] Johnson, W.-L. and Valente, A. Tactical language and culture training systems: Using artificial intelligence to teach foreign languages and cultures. In *Proc. IAAI 2008*, pages 1632–1639. AAAI Press, 2008. [cited at p. 68]

[81] Jones, E., Alexander, J., Andreou, A., Irani, P., and Subramanian, S. GesText: accelerometer-based gestural text-entry systems. In *Proc. CHI 2010*, pages 2173–2182. ACM, 2010. [cited at p. 64, 136, 139, 195]

[82] Kadobayashi, R., Nishimoto, K., and Mase, K. Design and evaluation of gesture interface of an immersive walk-through application for exploring cyberspace. In *Proc. Automatic Face and Gesture Recognition 1998*, pages 534–539. IEEE, 1998. [cited at p. 67, 81, 86]

[83] Kamal, A., Li, Y., and Lank, E. Teaching motion gestures via recognizer feedback. In *Proc. IUI 2014*, pages 73–82. ACM, 2014. [cited at p. 58, 60]

[84] Kang, S. K., Nam, M. Y., and Rhee, P.-K. Color based hand and finger detection technology for user interaction. In *Proc. ICHIT 2008*, pages 229–236. IEEE, 2008. [cited at p. 147]

[85] Karam, M. and Schraefel, M. C. A taxonomy of gestures in human computer interactions. Technical report, University of Southampton, 2005. [cited at p. 4]

[86] Kechavarzi, B., Sabanovic, S., and Weisman, K. Evaluation of control factors affecting the operator's immersion and performance in robotic teleoperation. In *Proc. RO-MAN 2012*, pages 608–613. IEEE, 2012. [cited at p. 94]

[87] Keller, C., Kühn, R., Engelbrecht, A., Korzetz, M., and Schlegel, T. A prototyping and evaluation framework for interactive ubiquitous systems. In Streitz, N. and Stephanidis, C., editors, *Distributed, Ambient, and Pervasive Interactions*, volume 8028 of *Lecture Notes in Computer Science*, pages 215–224. Springer Berlin Heidelberg, 2013. [cited at p. 44, 50]

[88] Kendon, A. How gestures can become like words. In Poyatos, F., editor, *Cross-cultural Perspectives in Nonverbal Communication*, pages 131–141. Hogrefe, 1988. [cited at p. 23]

[89] Khoshelham, K. and Elberink, S. O. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012. [cited at p. 2, 20, 42, 138]

[90] Kim, J., Mastnik, S., and André, E. EMG-based hand gesture recognition for realtime biosignal interfacing. In *Proc. IUI 2008*, pages 30–39. ACM, 2008. [cited at p. 55]

[91] Kipp, M., Neff, M., and Albrecht, I. An annotation scheme for conversational gestures: how to economically capture timing and form. *Language Resources and Evaluation*, 41(3-4):325–339, 2007. [cited at p. 26]

[92] Kistler, F. and André, E. Traveller: Eine interaktive virtuelle Umgebung für kulturelles Lernen, steuerbar mit benutzerdefinierten Ganzkörpergesten. In *Tagungsband Usability Day XI (uDay XI)*. FH Vorarlberg UCT Research, Austria, 2013. http://www.fhv.at/forschung/uct/uday/uday-11 (accessed 2015-9-15). [cited at p. 106]

[93] Kistler, F. and André, E. User-defined body gestures for an interactive storytelling scenario. In Kotzé, P., Marsden, G., Lindgaard, G., Wesson, J., and Winckler, M., editors, *Human-Computer Interaction – INTERACT*

2013, volume 8118 of *Lecture Notes in Computer Science*, pages 264–281. Springer Berlin Heidelberg, 2013. [cited at p. 91, 106]

[94] Kistler, F. and André, E. How can I interact? Comparing full body gesture visualizations. In *Proc. CHI PLAY 2015*, pages 583–588. ACM, 2015. [cited at p. 167]

[95] Kistler, F., André, E., Mascarenhas, S., Silva, A., Paiva, A., Degens, N., Hofstede, G., Krumhuber, E., Kappas, A., and Aylett, R. Traveller: An interactive cultural training system controlled by user-defined body gestures. In Kotzé, P., Marsden, G., Lindgaard, G., Wesson, J., and Winckler, M., editors, *Human-Computer Interaction – INTERACT 2013*, volume 8120 of *Lecture Notes in Computer Science*, pages 697–704. Springer Berlin Heidelberg, 2013. [cited at p. 106]

[96] Kistler, F., Endrass, B., and André, E. Full body interaction with virtual characters in an interactive storytelling scenario. In Bickmore, T., Marsella, S., and Sidner, C., editors, *Intelligent Virtual Agents*, volume 8637 of *Lecture Notes in Computer Science*, pages 236–239. Springer International Publishing, 2014. [cited at p. 106, 211]

[97] Kistler, F., Endrass, B., Damian, I., Dang, C. T., and André, E. Natural interaction with culturally adaptive virtual characters. *Journal on Multimodal User Interfaces*, 6:39–47, 2012. [cited at p. 39, 119, 178]

[98] Kistler, F., Sollfrank, D., Bee, N., and André, E. Full body gestures enhancing a game book for interactive story telling. In Si, M., Thue, D., André, E., Lester, J., Tanenbaum, J., and Zammitto, V., editors, *Interactive Storytelling*, volume 7069 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin / Heidelberg, 2011. [cited at p. 71, 73, 74, 119]

[99] Konda, K. R., Königs, A., Schulz, H., and Schulz, D. Real time interaction with mobile robots using hand gestures. In *Proc. HRI 2012*, pages 177–178. ACM, 2012. [cited at p. 70]

[100] Kopper, R., Silva, M. G., McMahan, R. P., and Bowman, D. Increasing the precision of distant pointing for large high-resolution displays. Technical report, Computer Science Virginia Tech, 2008. [cited at p. 138]

[101] Kratz, S. and Rohs, M. A $3 gesture recognizer: simple gesture recognition for devices equipped with 3D acceleration sensors. In *Proc. IUI 2010*, pages 341–344. ACM, 2010. [cited at p. 38]

[102] Kray, C., Nesbitt, D., Dawson, J., and Rohs, M. User-defined gestures for connecting mobile phones, public displays, and tabletops. In *Proc. MobileHCI 2010*, pages 239–248. ACM, 2010. [cited at p. 55]

[103] Kristensson, P. O. and Denby, L. C. Continuous recognition and visualization of pen strokes and touch-screen gestures. In *Proc. SBIM 2011*, pages 95–102. ACM, 2011. [cited at p. 34, 41, 60]

[104] Kristensson, P. O., Nicholson, T., and Quigley, A. Continuous recognition of one-handed and two-handed gestures using 3D full-body motion tracking sensors. In *Proc. IUI 2012*, pages 89–92. ACM, 2012. [cited at p. 34, 39]

[105] Kurdyukova, E., Redlin, M., and André, E. Studying user-defined ipad gestures for interaction in multi-display environment. In *Proc. IUI 2012*, pages 93–96. ACM, 2012. [cited at p. 55]

[106] Kurtenbach, G. and Hulteen, E. A. Gestures in human-computer communication. In Laurel, B. and Mountford, S. J., editors, *The Art of Human-Computer Interface Design*, pages 309–317. Addison-Wesley Longman Publishing Co., Inc., 1990. [cited at p. 4]

[107] Lambrecht, J., Kleinsorge, M., and Kruger, J. Markerless gesture-based motion control and programming of industrial robots. In *Proc. ETFA 2011*, pages 1–4. IEEE, 2011. [cited at p. 70]

[108] Langmann, B., Hartmann, K., and Loffeld, O. Depth camera technology comparison and performance evaluation. In *Proc. Pattern Recognition Applications and Methods 2012*, pages 438–444, 2012. [cited at p. 20]

[109] Lasseter, J. Principles of traditional animation applied to 3D computer animation. *SIGGRAPH Computer Graphics*, 21(4):35–44, 1987. [cited at p. 168]

[110] LaViola, J. J., Jr., Feliz, D. A., Keefe, D. F., and Zeleznik, R. C. Hands-free multi-scale navigation in virtual environments. In *Proc. I3D 2001*, pages 9–15. ACM, 2001. [cited at p. 67, 81]

[111] Li, Y. Protractor: A fast and accurate gesture recognizer. In *Proc. CHI 2010*, pages 2169–2172. ACM, 2010. [cited at p. 35, 36]

[112] Long, A. C., Jr., Landay, J. A., and Rowe, L. A. Implications for a gesture design tool. In *Proc. CHI 1999*, pages 40–47. ACM, 1999. [cited at p. 28, 29]

[113] Lucchese, G., Field, M., Ho, J., Gutierrez-Osuna, R., and Hammond, T. Gesturecommander: Continuous touch-based gesture prediction. In *Proc. CHI EA 2012*, pages 1925–1930. ACM, 2012. [cited at p. 32]

[114] MacKenzie, I. S. and Zhang, S. X. The design and evaluation of a high-performance soft keyboard. In *Proc. SIGCHI 1999*, pages 25–31. ACM, 1999. [cited at p. 136]

[115] Majaranta, P., Ahola, U.-K., and Špakov, O. Fast gaze typing with an adjustable dwell time. In *Proc. CHI 2009*, pages 357–360. ACM, 2009. [cited at p. 53]

[116] Mankoff, J. and Abowd, G. D. Cirrin: a word-level unistroke keyboard for pen input. In *Proc. UIST 1998*, pages 213–214. ACM, 1998. [cited at p. 53, 65, 136]

[117] Markussen, A., Jakobsen, M., and Hornbæk, K. Selection-based mid-air text entry on large displays. In Kotzé, P., Marsden, G., Lindgaard, G., Wesson, J., and Winckler, M., editors, *Human-Computer Interaction – INTERACT 2013*, volume 8117 of *Lecture Notes in Computer Science*, pages 401–418. Springer Berlin Heidelberg, 2013. [cited at p. 54, 65, 195]

[118] Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies. In *Proc. UIST 2011*, pages 315–326. ACM, 2011. [cited at p. 42, 50]

[119] Mascarenhas, S. F., Silva, A., Paiva, A., Aylett, R., Kistler, F., André, E., Degens, N., Hofstede, G. J., and Kappas, A. Traveller: an intercultural training system with intelligent agents. In *Proc. AAMAS 2013*, pages 1387–1388. International Foundation for Autonomous Agents and Multiagent Systems, 2013. [cited at p. 106]

[120] Mateas, M. and Stern, A. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In *Proc. Game Developers Conference: Game Design Track*, 2003. [cited at p. 67, 69]

[121] McNeill, D. So you think gestures are nonverbal? *Psychological Review*, 92(3):350–371, 1985. [cited at p. 7, 23, 26, 92]

[122] McNeill, D. *Head and Mind: What Gestures Reveal About Thought*. University of Chicago Press, 1992. [cited at p. 24, 41, 91, 102, 111]

[123] Medlock, M. C., Wixon, D., Terrano, M., Romero, R., and Fulton, B. Using the RITE method to improve products: A definition and a case study. *Usability Professionals Association*, 51, 2002. [cited at p. 218]

[124] Mehlmann, G., Gebhard, P., Endrass, B., and André, E. SceneMaker: Visual authoring of dialogue processes. In *Proc. Workshop on Knowledge and Reasoning in Practical Dialogue Systems 2011*, 2011. [cited at p. 73]

[125] Mitra, S. and Acharya, T. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(3):311–324, 2007. [cited at p. 4, 28]

[126] Mugellini, E., Sokhn, M., Carrino, S., and Abou Khaled, O. WiiNote: multimodal application facilitating multi-user photo annotation activity. In *Proc. ICMI-MLMI 2009*, pages 237–238. ACM, 2009. [cited at p. 64]

[127] Müller, J., Bailly, G., Bossuyt, T., and Hillgren, N. Mirrortouch: Combining touch and mid-air gestures for public displays. In *Proc. MobileHCI 2014*, pages 319–328. ACM, 2014. [cited at p. 5]

[128] Myers, C. S. and Rabiner, L. R. A comparative study of several dynamic time-warping algorithms for connected-word recognition. *The Bell System Technical Journal*, 60(7):1389–1409, 1981. [cited at p. 33, 43, 126, 154]

[129] Natarajan, P. and Nevatia, R. Online, real-time tracking and recognition of human actions. In *Proc. WMVC 2008*, pages 1–8. IEEE Computer Society, 2008. [cited at p. 31]

[130] Nebeling, M., Teunissen, E., Husmann, M., and Norrie, M. C. XDKinect: Development framework for cross-device interaction using Kinect. In *Proc. EICS 2014*, pages 65–74. ACM, 2014. [cited at p. 46, 50, 219]

[131] Nguyen-Duc-Thanh, N., Stonier, D., Lee, S. Y., and Kim, D.-H. A new approach for human-robot interaction using human body language. In *Proc. ICHIT 2011*, pages 762–769. Springer-Verlag, 2011. [cited at p. 70]

[132] Ni, T., Bowman, D., and North, C. AirStroke: bringing unistroke text entry to freehand gesture interfaces. In *Proc. CHI 2011*, pages 2473–2476. ACM, 2011. [cited at p. 39, 195]

[133] Nielsen, M., Störring, M., Moeslund, T., and Granum, E. A procedure for developing intuitive and ergonomic gesture interfaces for hci. In Camurri, A. and Volpe, G., editors, *Gesture-Based Communication in Human-Computer Interaction*, volume 2915 of *Lecture Notes in Computer Science*, pages 409–420. Springer Berlin Heidelberg, 2004. [cited at p. 54]

[134] Norman, D. A. and Draper, S. W. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., 1986. [cited at p. 54]

[135] Obaid, M., Häring, M., Kistler, F., Bühling, R., and André, E. User-defined body gestures for navigational control of a humanoid robot. In Ge, S., Khatib, O., Cabibihan, J.-J., Simmons, R., and Williams, M.-A., editors, *Social Robotics*, volume 7621 of *Lecture Notes in Computer Science*, pages 367–377. Springer Berlin Heidelberg, 2012. [cited at p. 94]

[136] Obaid, M., Kistler, F., Häring, M., Bühling, R., and André, E. A framework for user-defined body gestures to control a humanoid robot. *International Journal of Social Robotics*, pages 383–396, 2014. [cited at p. 197]

[137] Packard, E. *Die Insel der 1000 Gefahren*. Ravensburger Buchverlag, 2007. [cited at p. 72, 73, 74]

[138] Park, A.-Y. and Lee, S.-W. Gesture spotting in continuous whole body action sequences using discrete hidden markov models. In Gibet, S., Courty, N., and Kamp, J.-F., editors, *Gesture in Human-Computer Interaction and Simulation*, volume 3881 of *Lecture Notes in Computer Science*, pages 100–111. Springer Berlin Heidelberg, 2006. [cited at p. 31, 40]

[139] Pedersoli, F., Benini, S., Adami, N., and Leonardi, R. XKin: an open source framework for hand pose and gesture recognition using Kinect. *The Visual Computer*, pages 1–16, 2014. [cited at p. 46, 50, 157]

[140] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011. [cited at p. 48]

[141] Peiravi, A. and Taabbodi, B. A reliable 3D laser triangulation-based scanner with a new simple but accurate procedure for finding scanner parameters. *Journal of American Science*, 6(5):80–85, 2010. [cited at p. 21]

[142] Peng, X., Wang, L., Cai, Z., and Qiao, Y. Action and gesture temporal spotting with super vector representation. In Agapito, L., Bronstein, M. M., and Rother, C., editors, *Computer Vision - ECCV 2014 Workshops*, volume 8925 of *Lecture Notes in Computer Science*, pages 518–527. Springer International Publishing, 2015. [cited at p. 39]

[143] Perlin, K. Quikwriting: continuous stylus-based text entry. In *Proc. UIST 1998*, pages 215–216. ACM, 1998. [cited at p. 53, 65, 136, 137, 139]

[144] Piumsomboon, T., Clark, A., and Billinghurst, M. Physically-based interaction for tabletop augmented reality using a depth-sensing camera for environment mapping. In *Proc. IVCNZ 2011*, pages 161–166. ACM, 2011. [cited at p. 69]

[145] Portillo-Rodriguez, O., Sandoval-Gonzalez, O., Ruffaldi, E., Leonardi, R., Avizzano, C., and Bergamasco, M. Real-time gesture recognition, evaluation and feed-forward correction of a multimodal Tai-Chi platform. In Pirhonen, A. and Brewster, S., editors, *Haptic and Audio Interaction Design*, volume 5270 of *Lecture Notes in Computer Science*, pages 30–39. Springer Berlin Heidelberg, 2008. [cited at p. 59, 168]

[146] Ramey, A., González-Pacheco, V., and Salichs, M. A. Integration of a low-cost RGB-D sensor in a social robot for gesture recognition. In *Proc. HRI 2011*, pages 229–230. ACM, 2011. [cited at p. 29]

[147] Raybourn, E. M., Deagle, M. E., Mendini, K., and Heneghan, J. Adaptive thinking & leadership simulation game training for special forces officers. In *Proc. I/ITSEC 2005*. NTSA, 2005. [cited at p. 68]

[148] Ren, G. and O'Neill, E. Freehand gestural text entry for interactive tv. In *Proc. EuroITV 2013*, pages 121–130. ACM, 2013. [cited at p. 54, 66, 135, 139, 194, 195]

[149] Rich, C., Sidner, C. L., and Lesh, N. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, 2001. [cited at p. 160]

[150] Rigoll, G., Kosmala, A., and Eickeler, S. High performance real-time gesture recognition using hidden markov models. In Wachsmuth, I. and Fröhlich, M., editors, *Gesture and Sign Language in Human-Computer Interaction*, volume 1371 of *Lecture Notes in Computer Science*, pages 69–80. Springer Berlin Heidelberg, 1998. [cited at p. 32]

[151] Rubine, D. Specifying gestures by example. In *Proc. SIGGRAPH 1991*, pages 329–337. ACM, 1991. [cited at p. 28, 29, 36]

[152] Ruiz, J., Li, Y., and Lank, E. User-defined motion gestures for mobile interaction. In *Proc. CHI 2011*, pages 197–206. ACM, 2011. [cited at p. 55]

[153] Ruiz, J. and Vogel, D. Soft-constraints to reduce legacy and performance bias to elicit whole-body gestures with low arm fatigue. In *Proc. CHI 2015*, pages 3347–3350. ACM, 2015. [cited at p. 55]

[154] Saffer, D. *Designing Gestural Interfaces: Touchscreens and Interactive Devices*. O'Reilly Media, Inc., 2008. [cited at p. 54]

[155] Salem, M., Rohlfing, K., Kopp, S., and Joublin, F. A friendly gesture: Investigating the effect of multimodal robot behavior in human-robot interaction. In *Proc. RO-MAN 2011*, pages 247–252. IEEE, 2011. [cited at p. 25]

[156] Sato, E., Yamaguchi, T., and Harashima, F. Natural interface using pointing behavior for human-robot gestural interaction. *IEEE Transactions on Industrial Electronics*, 54(2):1105–1112, 2007. [cited at p. 70]

[157] Scherer, S., Marsella, S., Stratou, G., Xu, Y., Morbini, F., Egan, A., Rizzo, A., and Morency, L.-P. Perception markup language: Towards a standardized representation of perceived nonverbal behaviors. In Nakano, Y., Neff, M., Paiva, A., and Walker, M., editors, *Intelligent Virtual Agents*, volume

7502 of *Lecture Notes in Computer Science*, pages 455–463. Springer Berlin Heidelberg, 2012. [cited at p. 27]

[158] Schuler, D. and Namioka, A., editors. *Participatory Design: Principles and Practices*. L. Erlbaum Associates Inc., 1993. [cited at p. 54]

[159] Sharp, T., Keskin, C., Robertson, D., Taylor, J., Shotton, J., Kim, D., Rhemann, C., Leichter, I., Vinnikov, A., Wei, Y., Freedman, D., Kohli, P., Krupka, E., Fitzgibbon, A., and Izadi, S. Accurate, robust, and flexible real-time hand tracking. In *Proc. CHI 2015*, pages 3633–3642. ACM, 2015. [cited at p. 148]

[160] Shirwalkar, S., Singh, A., Sharma, K., and Singh, N. Telemanipulation of an industrial robotic arm using gesture recognition with Kinect. In *Proc. CARE 2013*, pages 1–6. IEEE, 2013. [cited at p. 70]

[161] Shoemaker, G., Findlater, L., Dawson, J. Q., and Booth, K. S. Mid-air text input techniques for very large wall displays. In *Proc. GI 2009*, pages 231–238. Canadian Information Processing Society, 2009. [cited at p. 54, 64, 136, 186, 190, 195]

[162] Sian, N., Yokoi, K., Kajita, S., Kanehiro, F., and Tanie, K. Whole body teleoperation of a humanoid robot – development of a simple master device using joysticks. In *Proc. Intelligent Robots and Systems 2002*, pages 2569–2574. IEEE, 2002. [cited at p. 94]

[163] Signer, B., Norrie, M. C., and Kurmann, U. iGesture: A Java framework for the development and deployment of stroke-based online gesture recognition algorithms. Technical Report TR561, ETH Zurich, 2007. [cited at p. 28]

[164] Sonnenburg, S., Rätsch, G., Henschel, S., Widmer, C., Behr, J., Zien, A., Bona, F. d., Binder, A., Gehl, C., and Franc, V. The SHOGUN machine learning toolbox. *The Journal of Machine Learning Research*, 11:1799–1802, 2010. [cited at p. 48]

[165] Spano, L. D., Cisternino, A., Paternò, F., and Fenu, G. GestIT: A declarative and compositional framework for multiplatform gesture definition. In *Proc. EICS 2013*, pages 187–196. ACM, 2013. [cited at p. 45, 50, 219]

[166] Stiefelhagen, R., Fugen, C., Gieselmann, R., Holzapfel, H., Nickel, K., and Waibel, A. Natural human-robot interaction using speech, head pose and gestures. In *Proc. IROS 2004*, pages 2422–2427, 2004. [cited at p. 70]

[167] Stipancic, T., Jerbic, B., Bucevic, A., and Curkovic, P. Programming an industrial robot by demonstration. In *Proc. DAAAM 2012*, pages 15–18. DAAAM International, 2012. [cited at p. 70]

[168] Su, X., Au, O. K.-C., and Lau, R. W. The implicit fan cursor: A velocity dependent area cursor. In *Proc. CHI 2014*, pages 753–762. ACM, 2014. [cited at p. 52, 138]

[169] Suma, E. A., Lange, B., Rizzo, A., Krum, D., and Bolas, M. FAAST: The flexible action and articulated skeleton toolkit. In *Proc. VR 2011*, pages 247–248. IEEE, 2011. [cited at p. 42]

[170] Suma, E. A., Krum, D. M., Lange, B., Koenig, S., Rizzo, A., and Bolas, M. Adapting user interfaces for gestural interaction with the flexible action and articulated skeleton toolkit. *Computers & Graphics*, 37(3):193–201, 2013. [cited at p. 42, 50, 157]

[171] Sussman, N. M. and Rosenfeld, H. M. Influence of culture, language, and sex on conversational distance. *Journal of Personality and Social Psychology*, 42:66–74, 1982. [cited at p. 178]

[172] Tan, N. *Posture and Space in Virtual Characters: Application to Ambient Interaction and Affective Interaction.* PhD thesis, Université Paris Sud, 2012. [cited at p. 26, 219]

[173] Tang, J. K. T. and Igarashi, T. CUBOD: A customized body gesture design tool for end users. In *Proc. BCS-HCI 2013*, pages 5:1–5:10. British Computer Society, 2013. [cited at p. 45, 50]

[174] Tang, R., Alizadeh, H., Tang, A., Bateman, S., and Jorge, J. A. Physio@Home: Design explorations to support movement guidance. In *Proc. CHI EA 2014*, pages 1651–1656. ACM, 2014. [cited at p. 62, 167, 168]

[175] Tang, R., Yang, X.-D., Bateman, S., Jorge, J., and Tang, A. Physio@Home: Exploring visual guidance and feedback techniques for physiotherapy exercises. In *Proc. CHI 2015*, pages 4123–4132. ACM, 2015. [cited at p. 62, 167]

[176] Tappert, C. C. Cursive script recognition by elastic matching. *IBM Journal of Research and Development*, 26(6):765–771, 1982. [cited at p. 33]

[177] Teófilo, L. F., Nogueira, P. A., and Silva, P. B. Gemini: A generic multimodal natural interface framework for videogames. In Rocha, Ã., Correia, A. M., Wilson, T., and Stroetmann, K. A., editors, *Advances in Information Systems and Technologies*, volume 206 of *Advances in Intelligent Systems and Computing*, pages 873–884. Springer Berlin Heidelberg, 2013. [cited at p. 44, 50]

[178] van der Sluis, I. and Krahmer, E. Generating multimodal references. *Discourse Processes*, 44(3):145–174, 2007. [cited at p. 67, 86]

[179] Velloso, E., Bulling, A., and Gellersen, H. AutoBAP: Automatic coding of body action and posture units from wearable sensors. In *Proc. ACII 2013*, pages 135–140. IEEE, 2013. [cited at p. 26]

[180] Velloso, E., Bulling, A., and Gellersen, H. MotionMA: Motion modelling and analysis by demonstration. In *Proc. CHI 2013*, pages 1309–1318. ACM, 2013. [cited at p. 62, 168]

[181] Vermeulen, J., Luyten, K., van den Hoven, E., and Coninx, K. Crossing the bridge over norman's gulf of execution: Revealing feedforward's true identity. In *Proc. CHI 2013*, pages 1931–1940. ACM, 2013. [cited at p. 59, 160, 161]

[182] Vogel, D. and Balakrishnan, R. Distant freehand pointing and clicking on very large, high resolution displays. In *Proc. UIST 2005*, pages 33–42. ACM, 2005. [cited at p. 51, 52, 137]

[183] Wagner, J., Lingenfelser, F., Baur, T., Damian, I., Kistler, F., and André, E. The Social Signal Interpretation (SSI) framework: Multimodal signal processing and recognition in real-time. In *Proc. MM 2013*, pages 831–834. ACM, 2013. [cited at p. 132]

[184] Walter, R., Bailly, G., and Müller, J. StrikeAPose: Revealing mid-air gestures on public displays. In *Proc. CHI 2013*, pages 841–850. ACM, 2013. [cited at p. 60, 167, 168]

[185] Watson, O. *Proxemic behavior: A cross-cultural study.* Mouton De Gruyter, 1970. [cited at p. 178]

[186] Wensveen, S. A. G., Djajadiningrat, J. P., and Overbeeke, C. J. Interaction frogger: A design framework to couple action and function through feedback and feedforward. In *Proc. DIS 2004*, pages 177–184. ACM, 2004. [cited at p. 58]

[187] Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., and Sloetjes, H. ELAN: a professional framework for multimodality research. In *Proc. LREC 2006*, 2006. http://tla.mpi.nl/tools/tla-tools/elan (accessed 2015-9-15); Max Planck Institute for Psycholinguistics, The Language Archive, Nijmegen, The Netherlands. [cited at p. 99, 110, 173]

[188] Wobbrock, J. O., Aung, H. H., Rothrock, B., and Myers, B. A. Maximizing the guessability of symbolic input. In *Proc. CHI EA 2005*, pages 1869–1872. ACM, 2005. [cited at p. 57]

[189] Wobbrock, J. O., Morris, M. R., and Wilson, A. D. User-defined gestures for surface computing. In *Proc. CHI 2009*, pages 1083–1092. ACM, 2009. [cited at p. 10, 24, 25, 55, 56, 57, 89, 91, 106, 216]

[190] Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc UIST 2007*, pages 159–168. ACM, 2007. [cited at p. 34, 38, 126, 154]

[191] Wölfel, M. Kinetic Space – 3D-Gestenerkennung für Dich und mich. *Konturen 2012*, 31:58–63, 2012. [cited at p. 33, 43, 50, 157]

[192] Wren, C., Azarbayejani, A., Darrell, T., and Pentland, A. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997. [cited at p. 69]

[193] Wu, P. and Miller, C. Interactive phrasebook – conveying culture through etiquette. In *Proc. CATS 2010 on ITS 2010*, pages 47–55, 2010. [cited at p. 68]

[194] Yamato, J., Ohya, J., and Ishii, K. Recognizing human action in time-sequential images using hidden markov model. In *Proc. CVPR 1992*, pages 379–385. IEEE, 1992. [cited at p. 31]

[195] Zaczynski, M. and Whitehead, A. D. Establishing design guidelines in interactive exercise gaming: Preliminary data from two posing studies. In *Proc. CHI 2014*, pages 1875–1884. ACM, 2014. [cited at p. 58, 59, 63, 176]

[196] Zafrulla, Z., Brashear, H., Starner, T., Hamilton, H., and Presti, P. American sign language recognition with the Kinect. In *Proc. ICMI 2011*, pages 279–286. ACM, 2011. [cited at p. 168, 169]

[197] Zhai, S., Hunter, M., and Smith, B. A. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *Proc. UIST 2000*, pages 119–128. ACM, 2000. [cited at p. 136]

[198] Zhang, L., Huang, Q., Liu, Q., Liu, T., Li, D., and Lu, Y. A teleoperation system for a humanoid robot with multiple information feedback and operational modes. In *Proc. ROBIO 2005*, pages 290–294. IEEE, 2005. [cited at p. 94]

[199] Zhao, W., Feng, H., Lun, R., Espy, D., and Reinthal, M. A Kinect-based rehabilitation exercise monitoring and guidance system. In *Proc. ICSESS 2014*, pages 762–765. IEEE, 2014. [cited at p. 63]

[200] Zhu, C. and Sheng, W. Online hand gesture recognition using neural network based segmentation. In *Proc. IROS 2009*, pages 2415–2420. IEEE, 2009. [cited at p. 41]

[201] Zinnen, A. and Schiele, B. A new approach to enable gesture recognition in continuous data streams. In *Proc. ISWC 2008*, pages 33–40. IEEE, 2008. [cited at p. 41]

Appendices

Appendix A

The FUBI Gesture Definition Language

A.1 XML Scheme

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.hcm-lab.de" xmlns="http://www.hcm-lab.de">
  <xs:annotation>
    <xs:documentation xml:lang="EN">
      Gesture Definiton Scheme for the Full Body Interaction Framework (FUBI)
      http://hcm-lab.de/fubi.html
    </xs:documentation>
  </xs:annotation>

  <!--The base element-->
  <xs:element name="FubiRecognizers">
    <xs:complexType>
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="JointRelationRecognizer" type="JointRelationRecognizer"/>
        <xs:element name="JointOrientationRecognizer"
            type="JointOrientationRecognizer"/>
        <xs:element name="LinearMovementRecognizer"
            type="LinearMovementRecognizer"/>
        <xs:element name="AngularMovementRecognizer"
            type="AngularMovementRecognizer"/>
        <xs:element name="FingerCountRecognizer" type="FingerCountRecognizer"/>
        <xs:element name="TemplateRecognizer" type="TemplateRecognizer"/>
        <xs:element name="CombinationRecognizer" type="CombinationRecognizer"/>
      </xs:choice>
      <xs:attribute name="globalMinConfidence" type="confidence"/>
      <xs:attribute default="false" name="globalUseFilteredData" type="xs:boolean"/>
    </xs:complexType>
```

```
25  </xs:element>
26
27  <!--All child elements, i.e. the different recognizers-->
28  <xs:complexType name="JointRelationRecognizer">
29    <xs:sequence>
30      <xs:choice>
31        <xs:sequence>
32          <xs:choice>
33            <xs:element name="Joints" type="Joints"/>
34            <xs:sequence>
35              <xs:element name="Joint" type="Joint" minOccurs ="0" maxOccurs="1"/>
36              <xs:element name="HandJoints" type="HandJoints"/>
37            </xs:sequence>
38          </xs:choice>
39          <xs:choice>
40            <xs:sequence>
41              <xs:element name="MaxValues" type="Values"/>
42              <xs:element minOccurs="0" maxOccurs="1" name="MinValues"
                    type="Values"/>
43            </xs:sequence>
44            <xs:sequence>
45              <xs:element name="MinValues" type="Values"/>
46              <xs:element minOccurs="0" maxOccurs="1" name="MaxValues"
                    type="Values"/>
47            </xs:sequence>
48            <xs:element minOccurs="1" maxOccurs="unbounded" name="Relation"
                  type="Relation"/>
49          </xs:choice>
50          <xs:element minOccurs="0" maxOccurs="1" name="MiddleJoint"
                type="MiddleJoint"/>
51        </xs:sequence>
52        <xs:element name="MiddleJoint" type="MiddleJoint"/>
53      </xs:choice>
54      <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
55    </xs:sequence>
56    <xs:attribute name="name" type="xs:ID" use="required" />
57    <xs:attribute default="visible" name="visibility" type="visibility"/>
58    <xs:attribute default="false" name="useLocalPositions" type="xs:boolean"/>
59    <xs:attribute name="minConfidence" type="confidence"/>
60    <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
61    <xs:attribute default="millimeter" name="measuringUnit" type="measures"/>
62  </xs:complexType>
63  <xs:complexType name="JointOrientationRecognizer">
64    <xs:sequence>
65      <xs:choice>
66        <xs:sequence>
67          <xs:element name="Joint" type="Joint"/>
68          <xs:element name="HandJoint" type="HandJoint" minOccurs="0"
                maxOccurs="1"/>
69        </xs:sequence>
70        <xs:element name="HandJoint" type="HandJoint"/>
71      </xs:choice>
72      <xs:choice>
73        <xs:sequence>
74          <xs:element name="MaxDegrees" type="Degrees"/>
75          <xs:element minOccurs="0" maxOccurs="1" name="MinDegrees"
                type="Degrees"/>
76        </xs:sequence>
```

```
77        <xs:sequence>
78          <xs:element name="MinDegrees" type="Degrees"/>
79          <xs:element minOccurs="0" maxOccurs="1" name="MaxDegrees"
                type="Degrees"/>
80        </xs:sequence>
81        <xs:element name="Orientation" type="Orientation"/>
82      </xs:choice>
83      <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
84    </xs:sequence>
85    <xs:attribute name="name" type="xs:ID" use="required" />
86    <xs:attribute default="visible" name="visibility" type="visibility"/>
87    <xs:attribute default="true" name="useLocalOrientations" type="xs:boolean"/>
88    <xs:attribute name="minConfidence" type="confidence"/>
89    <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
90  </xs:complexType>
91  <xs:complexType name="Degrees">
92    <xs:attribute name="x" type="xs:float" />
93    <xs:attribute name="y" type="xs:float" />
94    <xs:attribute name="z" type="xs:float" />
95  </xs:complexType>
96  <xs:complexType name="Orientation">
97    <xs:attribute name="x" type="xs:float" use="required" />
98    <xs:attribute name="y" type="xs:float" use="required" />
99    <xs:attribute name="z" type="xs:float" use="required" />
100   <xs:attribute default="45.0" name="maxAngleDifference" type="xs:float" />
101 </xs:complexType>
102 <xs:complexType name="LinearMovementRecognizer">
103   <xs:sequence>
104     <xs:choice>
105       <xs:sequence>
106         <xs:element name="Joints" type="Joints"/>
107         <xs:element name="HandJoints" type="HandJoints" minOccurs="0"
                maxOccurs="1"/>
108       </xs:sequence>
109       <xs:element name="HandJoints" type="HandJoints"/>
110     </xs:choice>
111     <xs:choice minOccurs="0" maxOccurs="1">
112       <xs:element name="Direction" type="Orientation"/>
113       <xs:element name="BasicDirection" type="BasicDirection"/>
114     </xs:choice>
115     <xs:element minOccurs="0" maxOccurs="1" name="Speed" type="Speed"/>
116     <xs:element minOccurs="0" maxOccurs="1" name="Length" type="Length"/>
117     <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
118   </xs:sequence>
119   <xs:attribute name="name" type="xs:ID" use="required" />
120   <xs:attribute default="visible" name="visibility" type="visibility"/>
121   <xs:attribute default="false" name="useLocalPositions" type="xs:boolean"/>
122   <xs:attribute name="minConfidence" type="confidence" />
123   <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
124   <xs:attribute default="true" name="useOnlyCorrectDirectionComponent"
          type="xs:boolean"/>
125 </xs:complexType>
126 <xs:complexType name="AngularMovementRecognizer">
127   <xs:sequence>
128     <xs:choice>
129       <xs:sequence>
130         <xs:element name="Joint" type="Joint"/>
```

```
131           <xs:element name="HandJoint" type="HandJoint" minOccurs="0"
                  maxOccurs="1"/>
132         </xs:sequence>
133         <xs:element name="HandJoint" type="HandJoint"/>
134       </xs:choice>
135       <xs:choice>
136         <xs:sequence>
137           <xs:element name="MaxAngularVelocity" type="AngularVelocity"/>
138           <xs:element minOccurs="0" maxOccurs="1" name="MinAngularVelocity"
                  type="AngularVelocity"/>
139         </xs:sequence>
140         <xs:sequence>
141           <xs:element name="MinAngularVelocity" type="AngularVelocity"/>
142           <xs:element minOccurs="0" maxOccurs="1" name="MaxAngularVelocity"
                  type="AngularVelocity"/>
143         </xs:sequence>
144         <xs:element minOccurs="1" maxOccurs="unbounded"
              name="BasicAngularVelocity" type="BasicAngularVelocity"/>
145       </xs:choice>
146       <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
147     </xs:sequence>
148     <xs:attribute name="name" type="xs:ID" use="required" />
149     <xs:attribute default="visible" name="visibility" type="visibility"/>
150     <xs:attribute name="minConfidence" type="xs:string" />
151     <xs:attribute default="true" name="useLocalOrientations" type="xs:boolean"/>
152     <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
153   </xs:complexType>
154   <xs:complexType name="FingerCountRecognizer">
155     <xs:sequence>
156       <xs:element minOccurs="0" maxOccurs="1" name="Joint" type="Joint"/>
157       <xs:element name="FingerCount" type="FingerCount"/>
158       <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
159     </xs:sequence>
160     <xs:attribute name="name" type="xs:ID" use="required" />
161     <xs:attribute default="visible" name="visibility" type="visibility"/>
162     <xs:attribute name="minConfidence" type="confidence" />
163     <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
164   </xs:complexType>
165   <xs:complexType name="TemplateRecognizer">
166     <xs:sequence>
167       <xs:choice>
168         <xs:element name="Joints" type="Joints" minOccurs="1"
              maxOccurs="unbounded"/>
169         <xs:sequence>
170           <xs:element name="Joint" type="Joint" minOccurs="0" maxOccurs="1"/>
171           <xs:element name="HandJoints" type="HandJoints" minOccurs="1"
                  maxOccurs="unbounded"/>
172         </xs:sequence>
173       </xs:choice>
174       <xs:element name="TrainingData" type="TrainingData" minOccurs="1"
            maxOccurs="unbounded"/>
175       <xs:element name="IgnoreAxes" type="IgnoreAxes" minOccurs="0"
            maxOccurs="1"/>
176       <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
            type="METAINFO"/>
177     </xs:sequence>
178     <xs:attribute name="name" type="xs:ID" use="required" />
```

```
179     <xs:attribute default="visible" name="visibility" type="visibility"/>
180     <xs:attribute name="minConfidence" type="confidence" />
181     <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
182     <xs:attribute default="false" name="useLocalTransformations" type="xs:boolean"/>
183     <xs:attribute default="millimeter" name="measuringUnit" type="measures"/>
184     <xs:attribute default="false" name="useOrientations" type="xs:boolean"/>
185     <xs:attribute default="true" name="useDTW" type="xs:boolean"/>
186     <xs:attribute default="0.5" name="maxWarpingFactor" type="xs:float"/>
187     <xs:attribute default="None" name="resamplingTechnique">
188       <xs:simpleType>
189         <xs:restriction base="xs:NMTOKEN">
190           <xs:enumeration value="None"/>
191           <xs:enumeration value="EquiDistant"/>
192           <xs:enumeration value="HermiteSpline"/>
193           <xs:enumeration value="PolyLine"/>
194         </xs:restriction>
195       </xs:simpleType>
196     </xs:attribute>
197     <xs:attribute default="-1" name="resampleSize" type="xs:int"/>
198     <xs:attribute default="GMR" name="stochasticModel">
199       <xs:simpleType>
200         <xs:restriction base="xs:NMTOKEN">
201           <xs:enumeration value="none"/>
202           <xs:enumeration value="GMR"/>
203           <!--<xs:enumeration value="HMM"/>-->
204         </xs:restriction>
205       </xs:simpleType>
206     </xs:attribute>
207     <xs:attribute default="5" name="numGMRStates" type="xs:unsignedInt"/>
208     <xs:attribute name="maxDistance" type="xs:float" use="required"/>
209     <xs:attribute name="distanceMeasure" default="euclidean">
210       <xs:simpleType>
211         <xs:restriction base="xs:NMTOKEN">
212           <xs:enumeration value="euclidean"/>
213           <xs:enumeration value="manhattan"/>
214           <xs:enumeration value="malhanobis"/>
215           <xs:enumeration value="turningAngleDiff"/>
216         </xs:restriction>
217       </xs:simpleType>
218     </xs:attribute>
219     <xs:attribute name="maxRotation" type="xs:float" default="45"/>
220     <xs:attribute name="aspectInvariant" type="xs:boolean" default="false"/>
221     <xs:attribute name="searchBestInputLength" type="xs:boolean" default="false"/>
222   </xs:complexType>
223   <xs:complexType name="CombinationRecognizer">
224     <xs:sequence>
225       <xs:element minOccurs="1" maxOccurs="unbounded" name="State" type="State"/>
226       <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
              type="METAINFO"/>
227     </xs:sequence>
228     <xs:attribute name="name" type="xs:ID" use="required" />
229     <xs:attribute default="false" name="waitUntilLastStateRecognizersStop"
          type="xs:boolean"/>
230   </xs:complexType>
231
232   <!-- Simple type definitions-->
233   <xs:simpleType name="confidence">
234     <xs:restriction base="xs:float">
235       <xs:minInclusive value="0"/>
```

```
236       <xs:maxInclusive value="1"/>
237     </xs:restriction>
238   </xs:simpleType>
239   <xs:simpleType name="visibility">
240     <xs:restriction base="xs:NMTOKEN">
241       <xs:enumeration value="visible" />
242       <xs:enumeration value="hidden" />
243     </xs:restriction>
244   </xs:simpleType>
245   <xs:simpleType name="joint">
246     <xs:restriction base="xs:NMTOKEN">
247       <xs:enumeration value="head" />
248       <xs:enumeration value="neck" />
249       <xs:enumeration value="torso" />
250       <xs:enumeration value="waist" />
251       <xs:enumeration value="leftShoulder" />
252       <xs:enumeration value="leftElbow" />
253       <xs:enumeration value="leftWrist" />
254       <xs:enumeration value="leftHand" />
255       <xs:enumeration value="rightShoulder" />
256       <xs:enumeration value="rightElbow" />
257       <xs:enumeration value="rightWrist" />
258       <xs:enumeration value="rightHand" />
259       <xs:enumeration value="leftHip" />
260       <xs:enumeration value="leftKnee" />
261       <xs:enumeration value="leftAnkle" />
262       <xs:enumeration value="leftFoot" />
263       <xs:enumeration value="rightHip" />
264       <xs:enumeration value="rightKnee" />
265       <xs:enumeration value="rightAnkle" />
266       <xs:enumeration value="rightFoot" />
267       <xs:enumeration value="faceNose" />
268       <xs:enumeration value="faceLeftEar" />
269       <xs:enumeration value="faceRightEar" />
270       <xs:enumeration value="faceForeHead" />
271       <xs:enumeration value="faceChin" />
272     </xs:restriction>
273   </xs:simpleType>
274   <xs:simpleType name="measures">
275     <xs:restriction base="xs:NMTOKEN">
276       <xs:enumeration value="millimeter" />
277       <xs:enumeration value="bodyHeight" />
278       <xs:enumeration value="torsoHeight" />
279       <xs:enumeration value="shoulderWidth" />
280       <xs:enumeration value="hipWidth" />
281       <xs:enumeration value="armLength" />
282       <xs:enumeration value="upperArmLength" />
283       <xs:enumeration value="lowerArmLength" />
284       <xs:enumeration value="legLength" />
285       <xs:enumeration value="upperLegLength" />
286       <xs:enumeration value="lowerLegLength" />
287     </xs:restriction>
288   </xs:simpleType>
289   <xs:simpleType name="handJoint">
290     <xs:restriction base="xs:NMTOKEN">
291       <xs:enumeration value="palm" />
292       <xs:enumeration value="thumb" />
293       <xs:enumeration value="index" />
294       <xs:enumeration value="middle" />
```

```
295       <xs:enumeration value="ring" />
296       <xs:enumeration value="pinky" />
297     </xs:restriction>
298   </xs:simpleType>
299   <xs:simpleType name="fingerCount">
300     <xs:restriction base="xs:integer">
301       <xs:minInclusive value="0"/>
302       <xs:maxInclusive value="5"/>
303     </xs:restriction>
304   </xs:simpleType>
305
306 <!-- complex type definitions-->
307   <xs:complexType name="Joints">
308     <xs:attribute name="main" use="required" type="joint"/>
309     <xs:attribute name="relative" type="joint"/>
310   </xs:complexType>
311   <xs:complexType name="HandJoints">
312     <xs:attribute name="main" use="required" type="handJoint"/>
313     <xs:attribute name="relative" type="handJoint"/>
314   </xs:complexType>
315   <xs:complexType name="Values">
316     <xs:attribute name="x" type="xs:float" />
317     <xs:attribute name="y" type="xs:float" />
318     <xs:attribute name="z" type="xs:float" />
319     <xs:attribute name="dist" type="xs:float" />
320   </xs:complexType>
321   <xs:complexType name="Relation">
322     <xs:attribute name="type" use="required">
323       <xs:simpleType>
324         <xs:restriction base="xs:NMTOKEN">
325           <xs:enumeration value="inFrontOf" />
326           <xs:enumeration value="behind" />
327           <xs:enumeration value="leftOf" />
328           <xs:enumeration value="rightOf" />
329           <xs:enumeration value="above" />
330           <xs:enumeration value="below" />
331           <xs:enumeration value="apartOf" />
332         </xs:restriction>
333       </xs:simpleType>
334     </xs:attribute>
335     <xs:attribute name="min" type="xs:float" />
336     <xs:attribute name="max" type="xs:float" />
337   </xs:complexType>
338   <xs:complexType name="MiddleJoint">
339     <xs:sequence>
340       <xs:choice>
341         <xs:element name="Joint" type="Joint"/>
342         <xs:element name="HandJoint" type="HandJoint"/>
343       </xs:choice>
344       <xs:choice>
345         <xs:sequence>
346           <xs:element name="MaxValues" type="Values"/>
347           <xs:element minOccurs="0" maxOccurs="1" name="MinValues" type="Values"/>
348         </xs:sequence>
349         <xs:sequence>
350           <xs:element name="MinValues" type="Values"/>
351           <xs:element minOccurs="0" maxOccurs="1" name="MaxValues" type="Values"/>
352         </xs:sequence>
```

```
353       <xs:element minOccurs="1" maxOccurs="unbounded" name="Relation"
              type="Relation"/>
354     </xs:choice>
355   </xs:sequence>
356 </xs:complexType>
357 <xs:complexType name="Joint">
358   <xs:attribute name="name" use="required" type="joint"/>
359 </xs:complexType>
360 <xs:complexType name="HandJoint">
361   <xs:attribute name="name" use="required" type="handJoint"/>
362 </xs:complexType>
363 <xs:complexType name="BasicDirection">
364   <xs:attribute name="type" use="required">
365     <xs:simpleType>
366       <xs:restriction base="xs:NMTOKEN">
367         <xs:enumeration value="left" />
368         <xs:enumeration value="right" />
369         <xs:enumeration value="up" />
370         <xs:enumeration value="down" />
371         <xs:enumeration value="forward" />
372         <xs:enumeration value="backward" />
373         <xs:enumeration value="anyDirection" />
374       </xs:restriction>
375     </xs:simpleType>
376   </xs:attribute>
377   <xs:attribute default="45.0" name="maxAngleDifference" type="xs:float" />
378 </xs:complexType>
379 <xs:complexType name="Speed">
380   <xs:attribute name="min" type="xs:float" />
381   <xs:attribute name="max" type="xs:float" />
382 </xs:complexType>
383 <xs:complexType name="Length">
384   <xs:attribute name="min" type="xs:float" />
385   <xs:attribute name="max" type="xs:float" />
386   <xs:attribute default="millimeter" name="measuringUnit" type="measures"/>
387 </xs:complexType>
388 <xs:complexType name="AngularVelocity">
389   <xs:attribute name="x" type="xs:float" />
390   <xs:attribute name="y" type="xs:float" />
391   <xs:attribute name="z" type="xs:float" />
392 </xs:complexType>
393 <xs:complexType name="BasicAngularVelocity">
394   <xs:attribute name="type" use="required">
395     <xs:simpleType>
396       <xs:restriction base="xs:NMTOKEN">
397         <xs:enumeration value="rollLeft" />
398         <xs:enumeration value="rollRight" />
399         <xs:enumeration value="pitchUp" />
400         <xs:enumeration value="pitchDown" />
401         <xs:enumeration value="yawLeft" />
402         <xs:enumeration value="yawRight" />
403       </xs:restriction>
404     </xs:simpleType>
405   </xs:attribute>
406   <xs:attribute name="min" type="xs:float" />
407   <xs:attribute name="max" type="xs:float" />
408 </xs:complexType>
409 <xs:complexType name="FingerCount">
410   <xs:attribute default="0" name="min" type="fingerCount"/>
```

```
411     <xs:attribute default="5" name="max" type="fingerCount"/>
412     <xs:attribute default="false" name="useMedianCalculation" type="xs:boolean"/>
413     <xs:attribute default="10" name="medianWindowSize" type="xs:unsignedInt" />
414   </xs:complexType>
415   <xs:complexType name="TrainingData">
416     <xs:attribute name="file" type="xs:string"/>
417     <xs:attribute name="start" type="xs:int" default="0"/>
418     <xs:attribute name="end" type="xs:int" default="-1"/>
419   </xs:complexType>
420   <xs:complexType name="IgnoreAxes">
421     <xs:attribute name="x" type="xs:boolean" default="false"/>
422     <xs:attribute name="y" type="xs:boolean" default="false"/>
423     <xs:attribute name="z" type="xs:boolean" default="false"/>
424   </xs:complexType>
425   <xs:complexType name="Recognizer">
426     <xs:attribute name="name" type="xs:string" use="required" />
427     <xs:attribute name="minConfidence" type="confidence" />
428     <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
429     <xs:attribute default="false" name="ignoreOnTrackingError" type="xs:boolean"/>
430   </xs:complexType>
431   <xs:complexType name="NotRecognizer">
432     <xs:attribute name="name" type="xs:string" use="required" />
433     <xs:attribute name="minConfidence" type="confidence" />
434     <xs:attribute default="false" name="useFilteredData" type="xs:boolean"/>
435     <xs:attribute default="true" name="ignoreOnTrackingError" type="xs:boolean"/>
436   </xs:complexType>
437   <xs:complexType name="AlternativeRecognizers">
438     <xs:choice minOccurs="1" maxOccurs="unbounded">
439       <xs:element name="Recognizer" type="Recognizer"/>
440       <xs:element name="NotRecognizer" type="NotRecognizer"/>
441     </xs:choice>
442   </xs:complexType>
443   <xs:complexType name="State">
444     <xs:sequence>
445       <xs:choice minOccurs="1" maxOccurs="unbounded">
446         <xs:element name="Recognizer" type="Recognizer"/>
447         <xs:element name="NotRecognizer" type="Recognizer"/>
448       </xs:choice>
449       <xs:element minOccurs="0" maxOccurs="1" name="AlternativeRecognizers"
              type="AlternativeRecognizers"/>
450       <xs:element minOccurs="0" maxOccurs="1" name="METAINFO"
              type="METAINFO"/>
451     </xs:sequence>
452     <xs:attribute default="0" name="minDuration" type="xs:float" />
453     <xs:attribute default="-1" name="maxDuration" type="xs:float" />
454     <xs:attribute default="1" name="timeForTransition" type="xs:float" />
455     <xs:attribute name="maxInterruptionTime" type="xs:float" />
456     <xs:attribute default="false" name="noInterruptionBeforeMinDuration"
          type="xs:boolean"/>
457     <xs:attribute default="restart" name="onFail">
458       <xs:simpleType>
459         <xs:restriction base="xs:NMTOKEN">
460           <xs:enumeration value="restart" />
461           <xs:enumeration value="goBack" />
462         </xs:restriction>
463       </xs:simpleType>
464     </xs:attribute>
465   </xs:complexType>
466
```

```
467   <!-- META information -->
468   <xs:complexType name="METAINFO">
469     <xs:sequence minOccurs="0" maxOccurs="unbounded">
470       <xs:element name="Property" type="Property"/>
471     </xs:sequence>
472   </xs:complexType>
473   <xs:complexType name="Property">
474     <xs:attribute name="name" type="xs:string" use="required" />
475     <xs:attribute name="value" type="xs:string" use="required" />
476   </xs:complexType>
477 </xs:schema>
```

A.2 Sample Gesture Definitions XML

```
1  <?xml version="1.0" encoding="utf-8" standalone="no"?>
2  <FubiRecognizers globalMinConfidence="0.51" xmlns="http://www.hcm-lab.de"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.hcm-lab.de␣
       http://www.hcm-lab.de/downloads/FubiRecognizers.xsd">
3    <!-- Joint relation samples -->
4    <JointRelationRecognizer name="HandsTogether">
5      <Joints main="rightHand" relative="leftHand"/>
6      <MaxValues dist="200"/>
7    </JointRelationRecognizer>
8    <JointRelationRecognizer name="RightHandInHeadHeight">
9      <Joints main="rightHand" relative="head"/>
10     <MaxValues y="150"/>
11     <MinValues y="-150"/>
12   </JointRelationRecognizer>
13   <JointRelationRecognizer name="LeftHandInHeadHeight">
14     <Joints main="leftHand" relative="head"/>
15     <MaxValues y="150"/>
16     <MinValues y="-150"/>
17   </JointRelationRecognizer>
18   <!-- Using shoulder width measuring unit and "Relation" instead of
         "Min/MaxValues" -->
19   <JointRelationRecognizer name="RightHandFront" measuringUnit="shoulderWidth">
20     <Joints main="rightHand" relative="torso"/>
21     <Relation type="inFrontOf" min="1"/>
22   </JointRelationRecognizer>
23   <JointRelationRecognizer name="LeftHandFront" measuringUnit="shoulderWidth">
24     <Joints main="leftHand" relative="torso"/>
25     <Relation type="inFrontOf" min="1"/>
26   </JointRelationRecognizer>
27   <!-- Special joint relation with additional middle joint for pointing -->
28   <JointRelationRecognizer name="PointingLeft" measuringUnit="armLength">
29     <Joints main="leftHand" relative="leftShoulder"/>
30     <Relation type="inFrontOf" min="0.25"/>
31     <Relation type="below" max="0.5"/>
32     <MiddleJoint>
33       <Joint name="leftElbow"/>
34       <Relation type="apartOf" max="0.25"/>
35     </MiddleJoint>
36   </JointRelationRecognizer>
37
38   <!-- Joint orientation samples -->
39   <JointOrientationRecognizer name="LeanLeft">
```

```
40    <Joint name="torso"/>
41    <MinDegrees z="15"/>
42  </JointOrientationRecognizer>
43  <JointOrientationRecognizer name="LeanRight">
44    <Joint name="torso"/>
45    <MaxDegrees z="-15"/>
46  </JointOrientationRecognizer>
47  <JointOrientationRecognizer name="LeanFront">
48    <Joint name="torso"/>
49    <MaxDegrees x="-15"/>
50  </JointOrientationRecognizer>
51  <JointOrientationRecognizer name="LeanBack">
52    <Joint name="torso"/>
53    <MinDegrees x="15"/>
54  </JointOrientationRecognizer>
55
56  <!-- Finger count samples -->
57  <FingerCountRecognizer name="OpenFist">
58    <Joint name="rightHand"/>
59    <FingerCount min="3"/>
60  </FingerCountRecognizer>
61  <FingerCountRecognizer name="ClosedFist">
62    <Joint name="rightHand"/>
63    <FingerCount max="1"/>
64  </FingerCountRecognizer>
65
66  <!-- Linear movement samples -->
67  <LinearMovementRecognizer name="RightHandRight">
68    <Joints main="rightHand" relative="rightShoulder"/>
69    <BasicDirection type="right" maxAngleDifference="30"/>
70    <Speed min="1000"/>
71  </LinearMovementRecognizer>
72  <LinearMovementRecognizer name="RightHandLeftDown">
73    <Joints main="rightHand" relative="rightShoulder"/>
74    <Direction x="-1" y="-1" z="0" maxAngleDifference="30"/>
75    <Speed min="1000"/>
76  </LinearMovementRecognizer>
77  <!-- A special linear movement: keeping the hand still -->
78  <LinearMovementRecognizer name="RightHandStill">
79    <Joints main="rightHand" relative="rightShoulder"/>
80    <Speed min="0" max="300"/>
81  </LinearMovementRecognizer>
82  <LinearMovementRecognizer name="RightKneeUpFront">
83    <Joints main="rightKnee" relative="waist"/>
84    <Direction x="0" y="1" z="-1"/>
85    <Speed min="200"/>
86  </LinearMovementRecognizer>
87  <LinearMovementRecognizer name="LeftKneeUpFront">
88    <Joints main="leftKnee" relative="waist"/>
89    <Direction x="0" y="1" z="-1"/>
90    <Speed min="200"/>
91  </LinearMovementRecognizer>
92
93  <!-- Angular movement samples -->
94  <!-- filteredData used as face tracking can be more instable-->
95  <AngularMovementRecognizer name="HeadDown" useFilteredData="true">
96    <Joint name="head"/>
97    <BasicAngularVelocity type="pitchDown" min="15"/>
98  </AngularMovementRecognizer>
```

```
<AngularMovementRecognizer name="HeadUp" useFilteredData="true">
  <Joint name="head"/>
  <BasicAngularVelocity type="pitchUp" min="15"/>
</AngularMovementRecognizer>
<AngularMovementRecognizer name="HeadLeft" useFilteredData="true">
  <Joint name="head"/>
  <BasicAngularVelocity type="yawLeft" min="20"/>
</AngularMovementRecognizer>
<AngularMovementRecognizer name="HeadRight" useFilteredData="true">
  <Joint name="head"/>
  <BasicAngularVelocity type="yawRight" min="20"/>
</AngularMovementRecognizer>

<!-- Template samples (require training data in separate file) -->
<TemplateRecognizer name="TemplateHeadNod" maxDistance="0.22"
    distanceMeasure="euclidean" useOrientations="true">
  <Joints main="head" />
  <TrainingData file="trainingData/HeadNod.xml"/>
  <IgnoreAxes y="true" z="true" />
</TemplateRecognizer>
<TemplateRecognizer name="TemplateTreasureChest" maxDistance="0.3"
    distanceMeasure="manhattan" aspectInvariant="false">
  <Joints main="rightHand" />
  <Joints main="leftHand" />
  <TrainingData file="trainingData/TreasureChest.xml"/>
</TemplateRecognizer>

<!-- Combination samples referencing the previous (or predefined) recognizer-->
<!-- Adding duration to a predefined posture -->
<CombinationRecognizer name="ArmsCrossedShortly">
  <State minDuration="0.5">
    <!-- Lower confidence as there are always problems here -->
    <Recognizer name="ArmsCrossed" minConfidence="0.3"/>
  </State>
</CombinationRecognizer>
<!-- Combining two relations -->
<CombinationRecognizer name="BothHandsUp">
  <State minDuration="0.3">
    <Recognizer name="leftHandOverShoulder"/>
    <Recognizer name="rightHandOverShoulder"/>
  </State>
</CombinationRecognizer>
<!-- Same for three relations -->
<CombinationRecognizer name="HandsFrontTogether">
  <State minDuration="0.3">
    <Recognizer name="RightHandFront" minConfidence="0.2"/>
    <Recognizer name="LeftHandFront" minConfidence="0.2"/>
    <Recognizer name="HandsTogether" minConfidence="0.2"/>
  </State>
</CombinationRecognizer>
<!-- Adding some duration and a required stillness to a predefined gesture -->
<CombinationRecognizer name="pointLong">
  <State minDuration="1">
    <Recognizer name="pointingRight"/>
    <Recognizer name="RightHandStill"/>
  </State>
</CombinationRecognizer>
<!-- Alternative recognizers combine the recognizers of a state with OR
    instead of AND -->
```

```
<CombinationRecognizer name="AtLeastOneHandInHeadHeight">
  <State minDuration="0.1">
    <Recognizer name="RightHandInHeadHeight"/>
    <AlternativeRecognizers>
      <Recognizer name="LeftHandInHeadHeight"/>
    </AlternativeRecognizers>
  </State>
</CombinationRecognizer>
<!-- NOT recognizers negate their outcome-->
<CombinationRecognizer name="OnlyOneHandInHeadHeight">
  <State minDuration="0.1">
    <Recognizer name="RightHandInHeadHeight"/>
    <NotRecognizer name="LeftHandInHeadHeight"/>
    <AlternativeRecognizers>
      <Recognizer name="LeftHandInHeadHeight"/>
      <NotRecognizer name="RightHandInHeadHeight"/>
    </AlternativeRecognizers>
  </State>
</CombinationRecognizer>
<!-- Multiple states create a sequence of linear movements -->
<CombinationRecognizer name="Zorro">
  <State maxDuration="1.2" minDuration="0.05" timeForTransition="0.4">
    <Recognizer name="RightHandRight"/>
  </State>
  <State maxDuration="1.2" minDuration="0.1" timeForTransition="0.4">
    <Recognizer name="RightHandLeftDown"/>
  </State>
  <State minDuration="0.05">
    <Recognizer name="RightHandRight"/>
  </State>
</CombinationRecognizer>
<!-- Combining finger counts for grabbing -->
<CombinationRecognizer name="Grabbing">
  <State minDuration="0.5" >
    <!-- ignoreOnTrackingError used, so it still works if shoulder is lost -->
    <Recognizer name="RightHandFront" ignoreOnTrackingError="true"/>
  </State>
  <State minDuration="0.2" timeForTransition="1">
    <Recognizer name="ClosedFist"/>
    <Recognizer name="RightHandFront" ignoreOnTrackingError="true"/>
  </State>
  <State minDuration="0.1" maxDuration="2" timeForTransition="1">
    <Recognizer name="OpenFist"/>
  </State>
  <State minDuration="0.2">
    <Recognizer name="ClosedFist"/>
    <Recognizer name="RightHandFront" ignoreOnTrackingError="true"/>
  </State>
</CombinationRecognizer>
<!-- Combining angular movements for HeadNod-->
<CombinationRecognizer name="HeadNod">
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadUp"/>
  </State>
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadDown"/>
  </State>
  <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
    <Recognizer name="HeadUp"/>
```

```
214     </State>
215     <State minDuration="0.05">
216       <Recognizer name="HeadDown"/>
217     </State>
218   </CombinationRecognizer>
219   <!-- and HeadShake -->
220   <CombinationRecognizer name="HeadShake">
221     <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
222       <Recognizer name="HeadLeft"/>
223     </State>
224     <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
225       <Recognizer name="HeadRight"/>
226     </State>
227     <State minDuration="0.05" maxDuration="0.8" timeForTransition="0.5">
228       <Recognizer name="HeadLeft"/>
229     </State>
230     <State minDuration="0.05" >
231       <Recognizer name="HeadRight"/>
232     </State>
233   </CombinationRecognizer>
234   <!-- Combining linear movements for walk in place -->
235   <CombinationRecognizer name="WalkInPlace">
236     <State maxDuration="1" timeForTransition="1.5" maxInterruptionTime="0.2">
237       <Recognizer name="RightKneeUpFront"/>
238     </State>
239     <State maxDuration="1" timeForTransition="1.5" maxInterruptionTime="0.2">
240       <Recognizer name="LeftKneeUpFront"/>
241     </State>
242     <State maxDuration="1" timeForTransition="1.5" maxInterruptionTime="0.2">
243       <Recognizer name="RightKneeUpFront"/>
244     </State>
245     <State maxDuration="1" timeForTransition="1.5" maxInterruptionTime="0.2">
246       <Recognizer name="LeftKneeUpFront"/>
247     </State>
248   </CombinationRecognizer>
249
250   <!--Additional samples from Traveller and the gesture visualization study -->
251   <!-- Basic recognizers are marked hidden for avoiding floods of recognitions
          -->
252   <!-- Bowing -->
253   <JointRelationRecognizer name="rightHandBelowShoulder" visibility="hidden">
254     <Joints main="rightHand" relative="rightShoulder"/>
255     <MaxValues y="0"/>
256   </JointRelationRecognizer>
257   <JointRelationRecognizer name="leftHandBelowShoulder" visibility="hidden">
258     <Joints main="leftHand" relative="leftShoulder"/>
259     <MaxValues y="0" />
260   </JointRelationRecognizer>
261   <JointOrientationRecognizer name="torsoStraight" visibility="hidden">
262     <Joint name="torso"/>
263     <MinDegrees x="-15"/>
264     <MaxDegrees x="15"/>
265   </JointOrientationRecognizer>
266   <JointOrientationRecognizer name="Bow" visibility="hidden">
267     <Joint name="torso"/>
268     <MaxDegrees x="-20"/>
269     <MinDegrees x="-65"/>
270   </JointOrientationRecognizer>
271   <CombinationRecognizer name="SmallBowWithHandsDown">
```

```
272     <State minDuration="0.5" timeForTransition="0.1">
273       <Recognizer name="rightHandBelowShoulder" ignoreOnTrackingError="true"/>
274       <Recognizer name="leftHandBelowShoulder" ignoreOnTrackingError="true"/>
275       <Recognizer name="torsoStraight"/>
276     </State>
277     <State minDuration="0.5" maxDuration="1.5" timeForTransition="0.2">
278       <Recognizer name="rightHandBelowShoulder" ignoreOnTrackingError="true"/>
279       <Recognizer name="leftHandBelowShoulder" ignoreOnTrackingError="true"/>
280       <Recognizer name="Bow"/>
281     </State>
282     <State minDuration="0.5">
283       <Recognizer name="rightHandBelowShoulder" ignoreOnTrackingError="true"/>
284       <Recognizer name="leftHandBelowShoulder" ignoreOnTrackingError="true"/>
285       <Recognizer name="torsoStraight"/>
286     </State>
287   </CombinationRecognizer>
288   <!-- PushFront -->
289   <JointRelationRecognizer name="right_hand_front" measuringUnit="upperArmLength"
          visibility="hidden">
290     <Joints main="rightWrist" relative="torso"/>
291     <Relation type="inFrontOf" min="1.5"/>
292   </JointRelationRecognizer>
293   <JointRelationRecognizer name="left_hand_front" measuringUnit="upperArmLength"
          visibility="hidden">
294     <Joints main="leftWrist" relative="torso"/>
295     <Relation type="inFrontOf" min="1.5"/>
296   </JointRelationRecognizer>
297   <LinearMovementRecognizer name="right_hand_forward" visibility="hidden">
298     <Joints main="rightWrist" relative="torso"/>
299     <BasicDirection type="forward"/>
300     <Speed min="200"/>
301   </LinearMovementRecognizer>
302   <LinearMovementRecognizer name="left_hand_forward" visibility="hidden">
303     <Joints main="leftWrist" relative="torso"/>
304     <BasicDirection type="forward"/>
305     <Speed min="200"/>
306   </LinearMovementRecognizer>
307   <CombinationRecognizer name="PushFront">
308     <State minDuration="0.1" timeForTransition="1.5">
309       <Recognizer name="right_hand_forward"/>
310       <Recognizer name="right_hand_forward"/>
311     </State>
312     <State>
313       <Recognizer name="right_hand_front"/>
314       <Recognizer name="right_hand_front"/>
315     </State>
316   </CombinationRecognizer>
317   <!-- Hands out -->
318   <JointRelationRecognizer name="RightHandRightOfShoulder" visibility="hidden">
319     <Joints main="rightWrist" relative="rightShoulder"/>
320     <Relation type="rightOf" min="250"/>
321   </JointRelationRecognizer>
322   <JointRelationRecognizer name="LeftHandLeftOfShoulder" visibility="hidden">
323     <Joints main="leftWrist" relative="leftShoulder"/>
324     <Relation type="leftOf" min="250"/>
325   </JointRelationRecognizer>
326   <CombinationRecognizer name="HandsOut">
327     <State minDuration="0.3">
328       <Recognizer name="RightHandRightOfShoulder"/>
```

```
329       <Recognizer name="LeftHandLeftOfShoulder"/>
330     </State>
331   </CombinationRecognizer>
332   <!--TurnAway-->
333   <JointOrientationRecognizer name="TurnedLeft">
334     <Joint name="torso"/>
335     <MinDegrees y="35"/>
336   </JointOrientationRecognizer>
337   <JointOrientationRecognizer name="TurnedRight">
338     <Joint name="torso"/>
339     <MaxDegrees y="-35"/>
340   </JointOrientationRecognizer>
341   <CombinationRecognizer name="TurnAway">
342     <State minDuration="0.3">
343       <Recognizer name="TurnedLeft"/>
344       <AlternativeRecognizers>
345         <Recognizer name="TurnedRight"/>
346       </AlternativeRecognizers>
347     </State>
348   </CombinationRecognizer>
349   <!--Clap-->
350   <JointRelationRecognizer name="RightHandInFrontOfBody">
351     <Joints main="rightHand" relative="torso"/>
352     <Relation type="inFrontOf" min="100"/>
353   </JointRelationRecognizer>
354   <JointRelationRecognizer name="LeftHandInFrontOfBody">
355     <Joints main="leftHand" relative="torso"/>
356     <Relation type="inFrontOf" min="100"/>
357   </JointRelationRecognizer>
358   <LinearMovementRecognizer name="leftHandMovesRight">
359     <Joints main="leftHand" relative="torso"/>
360     <Direction x="1" y="0" z="0"/>
361     <Speed min="250"/>
362   </LinearMovementRecognizer>
363   <JointRelationRecognizer name="HandsClose">
364     <Joints main="rightHand" relative="leftHand"/>
365     <Relation type="apartOf" max="150"/>
366   </JointRelationRecognizer>
367   <JointRelationRecognizer name="HandsApart" measuringUnit="shoulderWidth">
368     <Joints main="rightHand" relative="leftHand"/>
369     <Relation type="apartOf" min="1"/>
370   </JointRelationRecognizer>
371   <CombinationRecognizer name="Clap">
372     <State>
373       <Recognizer name="RightHandInFrontOfBody"/>
374       <Recognizer name="LeftHandInFrontOfBody"/>
375       <Recognizer name="HandsApart"/>
376     </State>
377     <State>
378       <Recognizer name="leftHandMovesRight"/>
379       <Recognizer name="rightHandMovesLeft"/>
380     </State>
381     <State>
382       <Recognizer name="RightHandInFrontOfBody"/>
383       <Recognizer name="LeftHandInFrontOfBody"/>
384       <Recognizer name="HandsClose"/>
385     </State>
386     <State>
387       <Recognizer name="RightHandInFrontOfBody"/>
```

```
388       <Recognizer name="LeftHandInFrontOfBody"/>
389       <Recognizer name="HandsApart"/>
390     </State>
391     <State>
392       <Recognizer name="leftHandMovesRight"/>
393       <Recognizer name="rightHandMovesLeft"/>
394     </State>
395     <State>
396       <Recognizer name="RightHandInFrontOfBody"/>
397       <Recognizer name="LeftHandInFrontOfBody"/>
398       <Recognizer name="HandsClose"/>
399     </State>
400   </CombinationRecognizer>
401   <!-- Right hand waving with left hand down, but right hand up -->
402   <JointRelationRecognizer name="leftHandOverShoulder" visibility="hidden">
403     <Joints main="leftHand" relative="leftShoulder"/>
404     <Relation type="above" min="0"/>
405   </JointRelationRecognizer>
406   <JointRelationRecognizer name="rightHandOverShoulder" visibility="hidden">
407     <Joints main="rightHand" relative="rightShoulder"/>
408     <MinValues y="0"/>
409   </JointRelationRecognizer>
410   <LinearMovementRecognizer name="rightHandMovesRight" visibility="hidden">
411     <Joints main="rightHand" relative="rightShoulder"/>
412     <Direction x="1" y="0" z="0"/>
413     <Speed min="250"/>
414   </LinearMovementRecognizer>
415   <LinearMovementRecognizer name="rightHandMovesLeft" visibility="hidden">
416     <Joints main="rightHand" relative="rightShoulder"/>
417     <Direction x="-1" y="0" z="0"/>
418     <Speed min="250"/>
419   </LinearMovementRecognizer>
420   <CombinationRecognizer name="RightHandWavingAboveTheShoulder">
421     <State minDuration="0.1" maxDuration="1.2" timeForTransition="0.3">
422       <Recognizer name="rightHandOverShoulder"/>
423       <Recognizer name="rightHandMovesLeft"/>
424       <NotRecognizer name="leftHandOverShoulder"/>
425     </State>
426     <State minDuration="0.1" maxDuration="1.2" timeForTransition="0.3">
427       <Recognizer name="rightHandOverShoulder"/>
428       <Recognizer name="rightHandMovesRight"/>
429       <NotRecognizer name="leftHandOverShoulder"/>
430     </State>
431     <State minDuration="0.1" maxDuration="1.2" timeForTransition="0.3">
432       <Recognizer name="rightHandOverShoulder"/>
433       <Recognizer name="rightHandMovesLeft"/>
434       <NotRecognizer name="leftHandOverShoulder"/>
435     </State>
436     <State minDuration="0.1">
437       <Recognizer name="rightHandOverShoulder"/>
438       <Recognizer name="rightHandMovesRight"/>
439       <NotRecognizer name="leftHandOverShoulder"/>
440     </State>
441   </CombinationRecognizer>
442   <!-- Sit down -->
443   <JointOrientationRecognizer name="RightKneeBent" visibility="hidden">
444     <Joint name="rightKnee"/>
445     <MaxDegrees x="-30"/>
446   </JointOrientationRecognizer>
```

```
447  <JointOrientationRecognizer name="LeftKneeBent" visibility="hidden">
448    <Joint name="leftKnee"/>
449    <MaxDegrees x="-30"/>
450  </JointOrientationRecognizer>
451  <CombinationRecognizer name="SitDown">
452    <State minDuration="0.5" maxInterruptionTime="0.15">
453      <Recognizer name="RightKneeBent" minConfidence="0.75"
           ignoreOnTrackingError="true"/>
454      <Recognizer name="LeftKneeBent" minConfidence="0.75"
           ignoreOnTrackingError="true"/>
455    </State>
456  </CombinationRecognizer>
457  <!-- Step forward -->
458  <LinearMovementRecognizer name="standingStill" visibility="hidden">
459    <Joints main="torso"/>
460    <Speed max="500"/>
461  </LinearMovementRecognizer>
462  <LinearMovementRecognizer name="movingForward" visibility="hidden">
463    <Joints main="torso"/>
464    <Direction x="0" y="0" z="-1"/>
465    <Speed min="150"/>
466  </LinearMovementRecognizer>
467  <CombinationRecognizer name="StepForward">
468    <State minDuration="0.1" timeForTransition="0.7">
469      <Recognizer name="standingStill"/>
470    </State>
471    <State minDuration="0.5">
472      <Recognizer name="movingForward"/>
473    </State>
474  </CombinationRecognizer>
475  <!-- Tip on shoulder -->
476  <JointRelationRecognizer name="rightHandInUpperFrontOfShoulder"
       visibility="hidden">
477    <Joints main="rightHand" relative="rightShoulder"/>
478    <MinValues y="-200" x="-200"/>
479    <MaxValues y="300" z="-200" x="300"/>
480  </JointRelationRecognizer>
481  <LinearMovementRecognizer name="rightHandUp" visibility="hidden">
482    <Joints main="rightHand" relative="rightShoulder"/>
483    <Direction x="0" y="1" z="0"/>
484    <Speed min="150"/>
485  </LinearMovementRecognizer>
486  <LinearMovementRecognizer name="rightHandDown" visibility="hidden">
487    <Joints main="rightHand" relative="rightShoulder"/>
488    <Direction x="0" y="-1" z="0"/>
489    <Speed min="150"/>
490  </LinearMovementRecognizer>
491  <CombinationRecognizer name="TipOnShoulder">
492    <State minDuration="0.1">
493      <Recognizer name="rightHandInUpperFrontOfShoulder"/>
494    </State>
495    <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
496      <Recognizer name="rightHandInUpperFrontOfShoulder"/>
497      <Recognizer name="rightHandDown"/>
498    </State>
499    <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
500      <Recognizer name="rightHandInUpperFrontOfShoulder"/>
501      <Recognizer name="rightHandUp"/>
502    </State>
```

```
503     <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
504       <Recognizer name="rightHandInUpperFrontOfShoulder"/>
505       <Recognizer name="rightHandDown"/>
506     </State>
507     <State>
508       <Recognizer name="rightHandInUpperFrontOfShoulder"/>
509       <Recognizer name="rightHandUp"/>
510     </State>
511   </CombinationRecognizer>
512   <!-- Treasure chest using linear movement with length restrictions -->
513   <LinearMovementRecognizer name="RightHandRightTC" visibility="hidden">
514     <Joints main="rightWrist"/>
515     <BasicDirection type="right"/>
516     <Length min="1" measuringUnit="upperArmLength"/>
517   </LinearMovementRecognizer>
518   <LinearMovementRecognizer name="rightHandDownTC" visibility="hidden">
519     <Joints main="rightWrist"/>
520     <BasicDirection type="down"/>
521     <Length min="0.5" measuringUnit="upperArmLength"/>
522   </LinearMovementRecognizer>
523   <LinearMovementRecognizer name="rightHandLeftTC" visibility="hidden">
524     <Joints main="rightWrist"/>
525     <BasicDirection type="left"/>
526     <Length min="1" measuringUnit="upperArmLength"/>
527   </LinearMovementRecognizer>
528   <LinearMovementRecognizer name="leftHandLeftTC" visibility="hidden">
529     <Joints main="leftWrist"/>
530     <BasicDirection type="left"/>
531     <Length min="1" measuringUnit="upperArmLength"/>
532   </LinearMovementRecognizer>
533   <LinearMovementRecognizer name="leftHandDownTC" visibility="hidden">
534     <Joints main="leftWrist"/>
535     <BasicDirection type="down"/>
536     <Length min="0.5" measuringUnit="upperArmLength"/>
537   </LinearMovementRecognizer>
538   <LinearMovementRecognizer name="leftHandRightTC" visibility="hidden">
539     <Joints main="leftWrist"/>
540     <BasicDirection type="right"/>
541     <Length min="1" measuringUnit="upperArmLength"/>
542   </LinearMovementRecognizer>
543   <CombinationRecognizer name="TreasureChest">
544     <State maxDuration="2" minDuration="0.2" maxInterruptionTime="0.08"
          timeForTransition="1">
545       <Recognizer name="RightHandRightTC"/>
546       <Recognizer name="leftHandLeftTC"/>
547     </State>
548     <State maxDuration="2" minDuration="0.15" maxInterruptionTime="0.12"
          timeForTransition="1">
549       <Recognizer name="rightHandDownTC"/>
550       <Recognizer name="leftHandDownTC"/>
551     </State>
552     <State minDuration="0.15" maxInterruptionTime="0.12">
553       <Recognizer name="rightHandLeftTC"/>
554       <Recognizer name="leftHandRightTC"/>
555     </State>
556   </CombinationRecognizer>
557   <!-- Climbing hands -->
558   <LinearMovementRecognizer name="leftHandDownwards" visibility="hidden">
559     <Joints main="leftHand" relative="leftShoulder"/>
```

```
560     <Direction x="0" y="-1" z="0"/>
561     <Speed min="150"/>
562   </LinearMovementRecognizer>
563   <CombinationRecognizer name="ClimbingHands">
564     <State minDuration="0.1">
565       <Recognizer name="rightHandOverShoulder"/>
566     </State>
567     <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
568       <Recognizer name="rightHandDownwards"/>
569     </State>
570     <State minDuration="0.1">
571       <Recognizer name="leftHandOverShoulder"/>
572     </State>
573     <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
574       <Recognizer name="leftHandDownwards"/>
575     </State>
576     <State minDuration="0.1">
577       <Recognizer name="rightHandOverShoulder"/>
578     </State>
579     <State maxDuration="1" timeForTransition="0.5" maxInterruptionTime="0.2">
580       <Recognizer name="rightHandDownwards"/>
581     </State>
582     <State minDuration="0.1">
583       <Recognizer name="leftHandOverShoulder"/>
584     </State>
585     <State>
586       <Recognizer name="leftHandDownwards"/>
587     </State>
588   </CombinationRecognizer>
589   <!-- Hands down together -->
590   <JointRelationRecognizer name="RightHandDown" visibility="hidden">
591     <Joints main="rightHand" relative="waist"/>
592     <Relation type="below" min="-100"/>
593   </JointRelationRecognizer>
594   <JointRelationRecognizer name="LeftHandDown" visibility="hidden">
595     <Joints main="leftHand" relative="waist"/>
596     <Relation type="below" min="-100"/>
597   </JointRelationRecognizer>
598   <CombinationRecognizer name="HandsDownTogether">
599     <State minDuration="0.3">
600       <Recognizer name="RightHandDown" minConfidence="0.2"/>
601       <Recognizer name="LeftHandDown" minConfidence="0.2"/>
602       <Recognizer name="HandsTogether" minConfidence="0.2"/>
603     </State>
604   </CombinationRecognizer>
605   <!-- Hold leg -->
606   <JointRelationRecognizer minConfidence="0.25" name="rightHandCloseToKnee"
          useFilteredData="true" visibility="hidden">
607     <Joints main="rightHand" relative="rightKnee"/>
608     <Relation type="apartOf" max="250"/>
609   </JointRelationRecognizer>
610   <JointRelationRecognizer minConfidence="0.25" name="leftHandCloseToKnee"
          useFilteredData="true" visibility="hidden">
611     <Joints main="leftHand" relative="rightKnee"/>
612     <Relation type="apartOf" max="250"/>
613   </JointRelationRecognizer>
614   <CombinationRecognizer name="HoldLeg">
615     <State minDuration="0.3">
616       <Recognizer name="rightHandCloseToKnee"/>
```

```
617       <Recognizer name="leftHandCloseToKnee"/>
618     </State>
619   </CombinationRecognizer>
620
621   <!-- EXPO-standing scheme for standing leg postures (only FUBI compatible
          parts) as joint relations and orientations-->
622   <!-- Left/Right leg shapes -->
623   <JointRelationRecognizer name="LegShapeCrossed">
624     <Joints main="rightFoot" relative="leftFoot"/>
625     <MaxValues x="0"/>
626   </JointRelationRecognizer>
627   <JointRelationRecognizer name="LegShapeClosed">
628     <Joints main="rightFoot" relative="leftFoot"/>
629     <MinValues x="5"/>
630     <MaxValues x="150"/>
631   </JointRelationRecognizer>
632   <JointRelationRecognizer name="LegShapeNormal">
633     <Joints main="rightFoot" relative="leftFoot"/>
634     <MinValues x="151"/>
635     <MaxValues x="300"/>
636   </JointRelationRecognizer>
637   <JointRelationRecognizer name="LegShapeApart">
638     <Joints main="rightFoot" relative="leftFoot"/>
639     <MinValues x="301"/>
640   </JointRelationRecognizer>
641   <!-- Right knee shape -->
642   <JointOrientationRecognizer name="RightKneeShapeNormal">
643     <Joint name="rightKnee"/>
644     <MinDegrees x="-20"/>
645   </JointOrientationRecognizer>
646   <JointOrientationRecognizer name="RightKneeShapeSlightlyFlexed">
647     <Joint name="rightKnee"/>
648     <MaxDegrees x="-20.01"/>
649     <MinDegrees x="-40"/>
650   </JointOrientationRecognizer>
651   <JointOrientationRecognizer name="RightKneeShapeFlexed">
652     <Joint name="rightKnee"/>
653     <MaxDegrees x="-40.01"/>
654   </JointOrientationRecognizer>
655   <!-- Left knee shape -->
656   <JointOrientationRecognizer name="LeftKneeShapeNormal">
657     <Joint name="leftKnee"/>
658     <MinDegrees x="-20"/>
659   </JointOrientationRecognizer>
660   <JointOrientationRecognizer name="LeftKneeShapeSlightlyFlexed">
661     <Joint name="leftKnee"/>
662     <MaxDegrees x="-20.01"/>
663     <MinDegrees x="-40"/>
664   </JointOrientationRecognizer>
665   <JointOrientationRecognizer name="LeftKneeShapeFlexed">
666     <Joint name="leftKnee"/>
667     <MaxDegrees x="-40.01"/>
668   </JointOrientationRecognizer>
669   <!-- Right leg position -->
670   <JointRelationRecognizer name="RightLegPositionBack">
671     <Joints main="rightFoot" relative="rightHip"/>
672     <MinValues z="151"/>
673   </JointRelationRecognizer>
674   <JointRelationRecognizer name="RightLegPositionNormal">
```

```
675     <Joints main="rightFoot" relative="rightHip"/>
676     <MinValues z="-150"/>
677     <MaxValues z="150"/>
678   </JointRelationRecognizer>
679   <JointRelationRecognizer name="RightLegPositionForward">
680     <Joints main="rightFoot" relative="rightHip"/>
681     <MaxValues z="-149"/>
682   </JointRelationRecognizer>
683   <!-- Left leg position -->
684   <JointRelationRecognizer name="LeftLegPositionBack">
685     <Joints main="leftFoot" relative="leftHip"/>
686     <MinValues z="151"/>
687   </JointRelationRecognizer>
688   <JointRelationRecognizer name="LeftLegPositionNormal">
689     <Joints main="leftFoot" relative="leftHip"/>
690     <MinValues z="-150"/>
691     <MaxValues z="150"/>
692   </JointRelationRecognizer>
693   <JointRelationRecognizer name="LeftLegPositionForward">
694     <Joints main="leftFoot" relative="leftHip"/>
695     <MaxValues z="-149"/>
696   </JointRelationRecognizer>
697
698   <!-- Wobbrock's 1\$ gestures as TemplateRecognizers for the right hand with
          different options, but all "trained" with four recordings using Gaussian
          mixture regression-->
699   <TemplateRecognizer name="triangle" maxDistance="2" distanceMeasure="malhanobis"
          aspectInvariant="true" useDTW="true" maxRotation="60"
          resamplingTechnique="EquiDistant" stochasticModel="GMR" numGMRStates="3"
          searchBestInputLength="true">
700     <Joints main="rightHand" relative="torso"/>
701     <TrainingData file="trainingData/Triangle1.xml"/>
702     <TrainingData file="trainingData/Triangle2.xml"/>
703     <TrainingData file="trainingData/Triangle3.xml"/>
704     <TrainingData file="trainingData/Triangle4.xml"/>
705     <IgnoreAxes z="true"/>
706   </TemplateRecognizer>
707   <TemplateRecognizer name="x" maxDistance="2" distanceMeasure="malhanobis"
          aspectInvariant="true" useDTW="true" maxRotation="90"
          resamplingTechnique="EquiDistant" stochasticModel="GMR" numGMRStates="3"
          searchBestInputLength="true">
708     <Joints main="rightHand" relative="torso"/>
709     <TrainingData file="trainingData/X1.xml"/>
710     <TrainingData file="trainingData/X2.xml"/>
711     <TrainingData file="trainingData/X3.xml"/>
712     <TrainingData file="trainingData/X4.xml"/>
713     <IgnoreAxes z="true"/>
714   </TemplateRecognizer>
715   <TemplateRecognizer name="rectangle" maxDistance="1.5"
          distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
          maxRotation="90" resamplingTechnique="EquiDistant" stochasticModel="GMR"
          numGMRStates="4" searchBestInputLength="true">
716     <Joints main="rightHand" relative="torso"/>
717     <TrainingData file="trainingData/Rectangle1.xml"/>
718     <TrainingData file="trainingData/Rectangle2.xml"/>
719     <TrainingData file="trainingData/Rectangle3.xml"/>
720     <TrainingData file="trainingData/Rectangle4.xml"/>
721     <IgnoreAxes z="true"/>
722   </TemplateRecognizer>
```

```
723  <TemplateRecognizer name="circle" maxDistance="10"
         distanceMeasure="turningAngleDiff" aspectInvariant="true" useDTW="true"
         maxRotation="45" resamplingTechnique="HermiteSpline" resampleSize="48"
         stochasticModel="GMR" numGMRStates="24" searchBestInputLength="true"
         useFilteredData="true">
724    <Joints main="rightHand" relative="torso"/>
725    <TrainingData file="trainingData/Circle1.xml"/>
726    <TrainingData file="trainingData/Circle2.xml"/>
727    <TrainingData file="trainingData/Circle3.xml"/>
728    <TrainingData file="trainingData/Circle4.xml"/>
729    <IgnoreAxes z="true"/>
730  </TemplateRecognizer>
731  <TemplateRecognizer name="check" maxDistance="1.5" distanceMeasure="malhanobis"
         aspectInvariant="false" useDTW="true" maxRotation="30"
         resamplingTechnique="HermiteSpline" stochasticModel="GMR"
         numGMRStates="3" searchBestInputLength="true">
732    <Joints main="rightHand" relative="torso"/>
733    <TrainingData file="trainingData/Check1.xml"/>
734    <TrainingData file="trainingData/Check2.xml"/>
735    <TrainingData file="trainingData/Check3.xml"/>
736    <TrainingData file="trainingData/Check4.xml"/>
737    <IgnoreAxes z="true"/>
738  </TemplateRecognizer>
739  <TemplateRecognizer name="caret" maxDistance="2" distanceMeasure="malhanobis"
         aspectInvariant="true" useDTW="true" maxRotation="45"
         resamplingTechnique="EquiDistant" stochasticModel="GMR" numGMRStates="2"
         searchBestInputLength="true">
740    <Joints main="rightHand" relative="torso"/>
741    <TrainingData file="trainingData/Caret1.xml"/>
742    <TrainingData file="trainingData/Caret2.xml"/>
743    <TrainingData file="trainingData/Caret3.xml"/>
744    <TrainingData file="trainingData/Caret4.xml"/>
745    <IgnoreAxes z="true"/>
746  </TemplateRecognizer>
747  <TemplateRecognizer name="question" maxDistance="1.7"
         distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
         maxRotation="60" resamplingTechnique="HermiteSpline"
         stochasticModel="GMR" numGMRStates="12" searchBestInputLength="true"
         useFilteredData="true">
748    <Joints main="rightHand" relative="torso"/>
749    <TrainingData file="trainingData/Question1.xml"/>
750    <TrainingData file="trainingData/Question2.xml"/>
751    <TrainingData file="trainingData/Question3.xml"/>
752    <TrainingData file="trainingData/Question4.xml"/>
753    <IgnoreAxes z="true"/>
754  </TemplateRecognizer>
755  <TemplateRecognizer name="arrow" maxDistance="5" distanceMeasure="malhanobis"
         aspectInvariant="true" useDTW="true" maxRotation="90"
         resamplingTechnique="PolyLine" stochasticModel="GMR" numGMRStates="4"
         searchBestInputLength="true">
756    <Joints main="rightHand" relative="torso"/>
757    <TrainingData file="trainingData/Arrow1.xml" start="0" end="68"/>
758    <TrainingData file="trainingData/Arrow2.xml"/>
759    <TrainingData file="trainingData/Arrow3.xml"/>
760    <TrainingData file="trainingData/Arrow4.xml"/>
761    <IgnoreAxes z="true"/>
762  </TemplateRecognizer>
763  <TemplateRecognizer name="leftSquareBracket" maxDistance="1.5"
         distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
```

```
      maxRotation="30" resamplingTechnique="EquiDistant" stochasticModel="GMR"
      numGMRStates="3" searchBestInputLength="true">
764     <Joints main="rightHand" relative="torso"/>
765     <TrainingData file="trainingData/LeftSquareBracket1.xml"/>
766     <TrainingData file="trainingData/LeftSquareBracket2.xml"/>
767     <TrainingData file="trainingData/LeftSquareBracket3.xml"/>
768     <TrainingData file="trainingData/LeftSquareBracket4.xml"/>
769     <IgnoreAxes z="true"/>
770   </TemplateRecognizer>
771   <TemplateRecognizer name="rightSquareBracket" maxDistance="1.5"
      distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
      maxRotation="30" resamplingTechnique="EquiDistant" stochasticModel="GMR"
      numGMRStates="3" searchBestInputLength="true">
772     <Joints main="rightHand" relative="torso"/>
773     <TrainingData file="trainingData/rightSquareBracket1.xml"/>
774     <TrainingData file="trainingData/rightSquareBracket2.xml"/>
775     <TrainingData file="trainingData/rightSquareBracket3.xml"/>
776     <TrainingData file="trainingData/rightSquareBracket4.xml"/>
777     <IgnoreAxes z="true"/>
778   </TemplateRecognizer>
779   <TemplateRecognizer name="v" maxDistance="1.5" distanceMeasure="malhanobis"
      aspectInvariant="false" useDTW="true" maxRotation="30"
      resamplingTechnique="HermiteSpline" stochasticModel="GMR"
      numGMRStates="2" searchBestInputLength="true">
780     <Joints main="rightHand" relative="torso"/>
781     <TrainingData file="trainingData/V1.xml"/>
782     <TrainingData file="trainingData/V2.xml"/>
783     <TrainingData file="trainingData/V3.xml"/>
784     <TrainingData file="trainingData/V4.xml"/>
785     <IgnoreAxes z="true"/>
786   </TemplateRecognizer>
787   <TemplateRecognizer name="delete" maxDistance="2.5" distanceMeasure="malhanobis"
      aspectInvariant="true" useDTW="true" maxRotation="90"
      resamplingTechnique="EquiDistant" stochasticModel="GMR" numGMRStates="3"
      searchBestInputLength="true">
788     <Joints main="rightHand" relative="torso"/>
789     <TrainingData file="trainingData/Delete1.xml"/>
790     <TrainingData file="trainingData/Delete2.xml"/>
791     <TrainingData file="trainingData/Delete3.xml"/>
792     <TrainingData file="trainingData/Delete4.xml"/>
793     <IgnoreAxes z="true"/>
794   </TemplateRecognizer>
795   <TemplateRecognizer name="leftCurlyBracket" maxDistance="1.8"
      distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
      maxRotation="45" resamplingTechnique="HermiteSpline"
      stochasticModel="GMR" numGMRStates="6" searchBestInputLength="true">
796     <Joints main="rightHand" relative="torso"/>
797     <TrainingData file="trainingData/LeftCurlyBracket1.xml"/>
798     <TrainingData file="trainingData/LeftCurlyBracket2.xml"/>
799     <TrainingData file="trainingData/LeftCurlyBracket3.xml"/>
800     <TrainingData file="trainingData/LeftCurlyBracket4.xml"/>
801     <IgnoreAxes z="true"/>
802   </TemplateRecognizer>
803   <TemplateRecognizer name="rightCurlyBracket" maxDistance="1"
      distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
      maxRotation="45" resamplingTechnique="HermiteSpline"
      stochasticModel="GMR" numGMRStates="6" searchBestInputLength="true">
804     <Joints main="rightHand" relative="torso"/>
805     <TrainingData file="trainingData/rightCurlyBracket1.xml"/>
```

```
806     <TrainingData file="trainingData/rightCurlyBracket2.xml"/>
807     <TrainingData file="trainingData/rightCurlyBracket3.xml"/>
808     <TrainingData file="trainingData/rightCurlyBracket4.xml"/>
809     <IgnoreAxes z="true"/>
810   </TemplateRecognizer>
811   <TemplateRecognizer name="star" maxDistance="2" distanceMeasure="malhanobis"
          aspectInvariant="true" useDTW="true" maxRotation="180"
          resamplingTechnique="EquiDistant" stochasticModel="GMR" numGMRStates="5"
          searchBestInputLength="true">
812     <Joints main="rightHand" relative="torso"/>
813     <TrainingData file="trainingData/Star1.xml"/>
814     <TrainingData file="trainingData/Star2.xml"/>
815     <TrainingData file="trainingData/Star3.xml"/>
816     <TrainingData file="trainingData/Star4.xml"/>
817     <IgnoreAxes z="1"/>
818   </TemplateRecognizer>
819   <TemplateRecognizer name="pigtail" maxDistance="2" distanceMeasure="malhanobis"
          aspectInvariant="true" useDTW="true" maxRotation="90"
          resamplingTechnique="HermiteSpline" stochasticModel="GMR"
          numGMRStates="12" searchBestInputLength="true" useFilteredData="true">
820     <Joints main="rightHand" relative="torso"/>
821     <TrainingData file="trainingData/PigTail1.xml"/>
822     <TrainingData file="trainingData/PigTail2.xml"/>
823     <TrainingData file="trainingData/PigTail3.xml"/>
824     <TrainingData file="trainingData/PigTail4.xml"/>
825     <IgnoreAxes z="true"/>
826   </TemplateRecognizer>
827   <!-- On the \$1 website, the question mark was replaced by the zig-zag -->
828   <TemplateRecognizer name="zigZag" maxDistance="2.5"
          distanceMeasure="malhanobis" aspectInvariant="true" useDTW="true"
          maxRotation="60" resamplingTechnique="EquiDistant" stochasticModel="GMR"
          numGMRStates="5" searchBestInputLength="true">
829     <Joints main="rightHand" relative="torso"/>
830     <TrainingData file="trainingData/ZigZag1.xml"/>
831     <TrainingData file="trainingData/ZigZag2.xml"/>
832     <TrainingData file="trainingData/ZigZag3.xml"/>
833     <TrainingData file="trainingData/ZigZag4.xml"/>
834     <IgnoreAxes z="true"/>
835   </TemplateRecognizer>
836 </FubiRecognizers>
```